

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Jia, Yunzhe

Title:
Pattern Aided Explainable Machine Learning

Date:
2019

Persistent Link:
<https://hdl.handle.net/11343/238743>

Terms and Conditions:
Terms and Conditions: Copyright in works deposited in Minerva Access is retained by the copyright owner. The work may not be altered without permission from the copyright owner. Readers may only download, print and save electronic copies of whole works for their own personal non-commercial use. Any use that exceeds these limits requires permission from the copyright owner. Attribution is essential when quoting or paraphrasing from these works.

UNIVERSITY OF MELBOURNE

DOCTORAL THESIS

Pattern Aided Explainable Machine Learning

Author:

Yunzhe JIA

Supervisor:

Prof. James BAILEY

Prof. Kotagiri

RAMAMOHANARAO

Prof. Christopher LECKIE

*A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy*

in the

School of Computing and Information Systems

ORCID Identifier: 0000-0001-7376-5838

April 14, 2020

Declaration of Authorship

I, Yunzhe JIA, declare that this thesis titled, “Pattern Aided Explainable Machine Learning” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Preface

This thesis is an original work by Yunzhe Jia. The chapters are written based on the following publications and submissions:

- Chapter Four is based on the publication: Yunzhe Jia, James Bailey, Ramamohanarao Kotagiri, and Christopher Leckie. “Pattern based Feature Generation.” In: *Feature Engineering for Machine Learning and Data Analytics*, 2018, p.245.
- Chapter Five is based on the publication: Yunzhe Jia, James Bailey, Ramamohanarao Kotagiri, Christopher Leckie, and Xingjun Ma. “Exploiting frequent patterns to explain individual predictions”. In: *Knowledge and Information Systems*, 2019, pp.1-24.
- Chapter Six is based on the publication: Yunzhe Jia, James Bailey, Ramamohanarao Kotagiri, and Christopher Leckie. “Improving the quality of explanations with local embedding perturbations” In: *Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2019, pp.875-884.
- Chapter Seven is based on the submission: Yunzhe Jia, James Bailey, Ramamohanarao Kotagiri, and Christopher Leckie. “Learning with expert supervision to make reliable predictions”. In: *Proceedings of the 24th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2020 (submitted).
- Chapter Eight is based on the submission: Yunzhe Jia, James Bailey, Ramamohanarao Kotagiri, and Christopher Leckie. “A Joint Learning Approach for Classification and Pattern Based Explanations”. In: *Proceedings of SIAM International Conference on Data Mining*, 2020 (planned to be submitted).

Abstract

Interpretability has been recognized as an important property of machine learning models. Lack of interpretability brings challenges for the deployments of many black models such as random forest, support vector machine (SVM) and neural networks. One aspect of interpretability is the ability to provide explanations for the predictions of a model, and explanations help users to understand the logical reasoning behind a model, thus giving users greater confidence to accept/reject predictions. Explanations are useful and sometimes even mandatory in domains like medical analysis, marketing, and criminal investigations, where decisions based on the predictions may have severe consequences.

Traditional classifiers can be categorized into interpretable models (or white-box models) and non-interpretable models (or black-box models). Interpretable models are the models whose internal structures or parameters are simple and can be easily explained, the examples of interpretable models are decision trees, linear models, logistic regression models. Non-interpretable models are the models whose are complex and difficult to explain, the examples of non-interpretable models are random forests, support vector machines and neural networks. Though the white-box models are intrinsically easy to interpret, they usually fail to achieve comparative accuracy as the black-box models. To facilitate the successful deployments of machine learning models when both of interpretability and accuracy are desired, there exist two directions of research: (1) increasing the accuracy of white-box models, and (2) increasing the interpretability of black-box models.

Patterns are conjunctions of feature-value conditions, which are intrinsically easy to comprehend, and they have been shown to have good predictive power as well. The objectives of the thesis is to propose methods to utilize patterns to increase the accuracy of white-box models by interpretable feature engineering and building interpretable models, and help black-box models provide explanations.

First, we discuss the pattern based interpretable feature engineering. Pattern based feature engineering extracts patterns from data, selects the representative patterns from the extracted candidates and then projects the instances in the original feature space into new pattern based feature space with a mapping function. The new pattern based features can be more discriminative, and meanwhile they are interpretable.

Second, we propose a method to explain any classifier using contrast patterns. Given a model and a query instance being explained, the proposed method first generates a synthetic neighborhood around the query using random perturbations, then labels the synthetic instances using the model, finally the method mines contrast patterns from the synthetic neighborhood and selects top K patterns as the final explanations. The experiments show that the method is able to achieve high faithfulness such that the explanations truly reveal how a model “thinks”, moreover the method is able to support scenarios where there exist multiple possible explanations.

Third, we analyse why some instances are difficult to explain. We investigate the crucial process of generating synthetic neighbors for local explanations methods, as different synthetic neighbors can result in explanations of different quality, and in many cases, the random perturbation does work well. We analyze the relationship of local intrinsic dimensionality (LID) and the quality of explanations, and propose a LID based method

to generate the synthetic neighbors such that the generated synthetic neighbors are more effective than the ones generated by other baseline methods.

Then we propose an interpretable model that achieves comparable accuracy with the state of the art baselines using patterns. The proposed method is a pattern based partition-wise linear models method that can be trained together with expert explanations. It divides the data space into several partitions, and each partition is represented by a pattern and is associated with a local linear model. The overall prediction for a query is a linear combination of the local linear models in the activated partitions. The model is interpretable and is able to work with expert explanations as a loss function in terms of explanations is part of the ultimate loss function. The results show that the proposed method is able to make superior reliable predictions and achieve competitive accuracy comparing with the baseline methods.

Finally, we show how to construct a model to make both accurate and reliable predictions by jointly learning explanations and class labels using multi-task learning in neural networks. We propose a neural network structure that is able to jointly train the class labels and the explanations where the explanations can be treated as another label information. We fit a neural network in the framework of multi-task learning. The neural network starts with a set of shared layers and then split into two separate layers where one is for class label and the other is for explanations. The experiments suggest that the proposed method is able to make reliable predictions.

In summary, this work recognizes the importance of interpretability of machine learning models, and it utilizes patterns to help improve the interpretability of machine learning models through: interpretable feature generation, pattern based model-agnostic local explanation extraction, pattern based partition-wise linear models and joint learning framework with explanations and class labels. We also investigate why a particular instance is difficult to explain using local intrinsic dimensionality. All work is supported by theoretical analysis and empirical evaluations.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisors Prof. James Bailey, Prof. Kotagiri Ramamohanarao and Prof. Christopher Leckie for the continuous support of my Ph.D study and related research, for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis. I could not have imagined having better supervisors and mentors for my Ph.D study.

Besides my supervisors, I would like to thank my advisory committee chair Dr. Sean Maynard and research coordinator Prof. Tony Wirth for their insightful comments and encouragement, but also for the hard question which incited me to widen my research from various perspectives.

I thank my fellow labmates Shuo, Xingjun, Donia, Yang, Simone, Moha, Sobia for the stimulating discussions.

Last but not the least, I would like to thank my parents, to my love Yu and to her parents for supporting me spiritually throughout writing this thesis and my my life in general.

Contents

Declaration of Authorship	iii
Preface	v
Abstract	vii
Acknowledgements	ix
1 Introduction	1
1.1 What is interpretability?	1
1.1.1 Why does interpretability matter?	1
1.1.2 Trade-off between interpretability and accuracy	2
1.2 Explainability	2
1.2.1 Different between interpretability and Explainability	2
1.2.2 Types of explanations	3
1.2.3 How to get the explanations?	3
1.2.4 Improving performance of models using explanations	4
1.2.5 Properties of explanations	4
1.2.6 Decision making process with explanations	6
1.3 Relationship of physical world, machine learning model and human	6
1.4 A possible solution to improve explainability - patterns	7
1.4.1 How can patterns improve explainability of models?	8
1.4.2 Patterns as explanations versus other forms of explanations	8
1.5 Thesis structure	9
2 Background	11
2.1 Introduction	11
2.2 Basic concepts	11
2.2.1 Classification and pattern	11
2.3 Explanations	13
2.3.1 Representations of explanations	13
2.4 Evaluation of explanations	13
2.4.1 Metrics	13
2.4.2 Faithfulness and reliability	14
3 Literature Review	17
3.1 Introduction	17
3.2 White-box models	17
3.2.1 Linear models	17
3.2.2 Logistic regression	18
3.2.3 Decision tree	19
3.2.4 Naive Bayes Classifier	20
3.2.5 K-Nearest Neighbors	21

3.3	Model-agnostic explanation methods	21
3.3.1	Partial Dependence Plot	21
3.3.2	Individual Conditional Expectation	23
3.3.3	Feature Importance	24
3.3.4	Shapley Values	26
3.3.5	Local interpretable model-agnostic explanations	27
3.4	Example-Based Explanations	31
3.4.1	K-nearest neighbors	31
3.4.2	Adversarial examples	31
3.4.3	Counterfactual Explanations	33
3.4.4	Prototypes and Criticisms	35
3.5	Summary of explanation methods	37
4	Pattern based Interpretable Feature Generation	39
4.1	Introduction	39
4.2	Preliminaries	40
4.2.1	Patterns in transaction data	40
4.2.2	Connection with patterns in other format	41
4.3	Framework of pattern based feature generation	43
4.3.1	Pattern mining	44
4.3.2	Pattern selection	45
4.3.3	Feature generation	45
4.4	Pattern mining algorithms	45
4.4.1	Frequent pattern mining	46
4.4.2	Contrast pattern mining	48
4.5	Pattern selection approaches	49
4.5.1	Post-processing pruning	49
4.5.2	In-processing pruning	51
4.6	Pattern based feature generation	52
4.6.1	Unsupervised mapping functions	52
4.6.2	Supervised mapping functions	53
4.6.3	Feature generation for sequence data and graph data	55
4.6.4	Comparison with similar techniques	55
4.7	Pattern based feature generation for classification	56
4.7.1	Problem statement	56
4.7.2	Direct classification in the pattern space	56
4.7.3	Indirect classification in the pattern space	57
4.7.4	Connection with stacking technique	58
4.8	Pattern based feature generation for clustering	58
4.8.1	Clustering in the pattern space	58
4.8.2	Subspace clustering	59
4.9	Conclusion	60
5	Exploiting patterns to explain individual predictions	61
5.1	Introduction	61
5.2	Related Work	62
5.3	Patterns as Explanations	65
5.4	Proposed Algorithm: PALEX	65
5.4.1	Methodology	66
5.4.2	Sampling in the frequent pattern space	68
5.4.3	Explanation selection	69

5.4.4	Example	70
5.4.5	Complexity analysis	71
5.5	Experiments	72
5.5.1	Metrics	72
5.5.2	Experimental setup	72
5.5.3	Faithfulness to oracle on synthetic data	73
5.5.4	Faithfulness to decision trees on UCI data	74
5.5.5	Impact of sampling method	75
5.5.6	Impact of explanation selection	75
5.5.7	Impact of parameters	78
5.6	Limitations and Future Work	78
5.7	Conclusion	81
6	Improving the Quality of Explanations with Local Embedding Perturbations	83
6.1	Introduction	83
6.2	Related work	85
6.3	Preliminaries	85
6.4	Proposed Method	87
6.4.1	General framework of local explanations	87
6.4.2	Local perturbation operators	87
6.4.3	Learning local explanations	87
6.4.4	LEAP - Local embedding aided perturbation	89
6.4.5	Local explanation extraction with LEAP	90
6.5	Experiments	91
6.5.1	Explanation metrics	91
6.5.2	Settings	91
6.5.3	Synthetic data and true explanations	92
6.5.4	UCI data and true explanations	93
6.5.5	Evaluation on synthetic data	94
6.5.6	Evaluation on UCI data	94
6.5.7	Visualization of the quality of explanations for different methods	94
6.5.8	Locality analysis	97
6.5.9	Example: Why does the quality of explanations matter?	99
6.6	Conclusion	100
7	Learning with Expert Supervision to Make Reliable Predictions	101
7.1	Introduction	101
7.2	Related work	102
7.3	Preliminaries	103
7.4	Methodology	104
7.4.1	Model description	104
7.4.2	Model explanations of xPWL	105
7.4.3	Objective function	105
7.4.4	Loss functions	106
7.4.5	Convex relaxation	106
7.4.6	Solution	107
7.5	Algorithm	107
7.6	Experiments	109
7.6.1	Settings	109
7.6.2	Evaluation	110
7.7	Conclusion	113

8	Joint Learning for Classification and Pattern Based Explanations	117
8.1	Introduction	117
8.2	Related Work	119
8.2.1	Multi-task learning	119
8.3	Our Proposed Framework - ExplNet	120
8.3.1	Explanation unit.	120
8.3.2	Overview	120
8.3.3	ExplNet	120
8.3.4	Contrast patterns as explanations	121
8.3.5	Motivation example	122
8.4	Experiments	123
8.4.1	Datasets	123
8.4.2	Baseline methods	123
8.4.3	Experimental settings	124
8.4.4	Metrics	124
8.4.5	Evaluation with expert explanation from a single tree	124
8.4.6	Evaluation with expert patterns from committee of trees	125
8.4.7	What happens when ExplNet and other models disagree?	125
8.4.8	Impact of α	128
8.5	Conclusion	128
9	Conclusion and Future work	133
9.1	Conclusion	133
9.2	Future works	134
9.2.1	Collecting expert explanations	134
9.2.2	Using explanations to improve predictive accuracy	135
	Bibliography	137

List of Figures

1.1	Example of two accurate models based on different evidence	2
1.2	Trade-off between interpretability and accuracy	3
1.3	Example of a decision tree	4
1.4	Demonstration of decision making process	6
1.5	Relationship of reality, model and human	8
3.1	Partial dependency plots for the bicycle count prediction model and temperature, humidity and wind speed	22
3.2	Individual conditional expectation plot of cervical cancer probability by age	24
3.3	The importance for each of the features in predicting bike counts with a support vector machine.	25
3.4	Shapley values for a certain day	27
3.5	Example to present intuition for LIME	28
3.6	Example of LIME explanation on tabular data	29
3.7	Example of LIME explanation on text data	30
3.8	Making a panda look like a gibbon for a neural network	32
3.9	Demonstration of squared maximum mean discrepancy measure (MMD2) and different selections of prototypes.	36
4.1	Framework of pattern based generation	43
4.2	Framework of pattern based generation	44
4.3	Constructing a pattern space using binary mapping function	53
5.1	Overview of PALEX process	67
5.2	Comparison of explanations generated by LIME, aLIME and PALEX. . .	71
5.3	Impact of parameters on faithfulness (F_1 score)	80
6.1	Illustration of a ‘good’ neighborhood and ‘bad’ neighborhood (triangles and circles are different labels). Given a test instance (filled circle) and the underlying classification boundary (solid line), generating ‘good’ neighbors (left) helps to mimic a local model (dashed line) that is similar to the true local model, while a local model (right) from ‘bad’ neighbors is far away from the true model. LID (local intrinsic dimensionality) is a possible metric for the quality of the neighbors.	84
6.2	Demonstration of difficulty of explanation for different neighborhood generating methods. The black solid line is the classification boundary, and difficulty (explanation quality) varies from difficult (red) to easy (yellow). The average explanation quality is shown in the bottom-right of each figure.	97
6.3	Demonstration of the synthetic neighborhoods that are generated by different methods. The test instances are shown in blue color, and the synthetic neighborhoods are shown in orange color. The quality of explanations extracted from the synthetic neighbors is shown in the bottom-right.	98

6.4	Locality analysis (plot of the accuracy of the local models when increasing the radius)	99
7.1	Demonstration of xPWL	105
7.2	Impact of α . X-axis represents α and y-axis represents accuracy (left) and reliability (right).	112
7.3	Impact of size of <i>minSupp</i> . X-axis represents <i>minSupp</i> and y-axis represents accuracy (left) and reliability (right).	112
7.4	Impact of level of expert supervision. X-axis represents percentage of expert supervision and y-axis represents accuracy (left) and reliability (right).	113
8.1	An overview of ExplNet structure. x is the input instance, $L_1, \dots, L_n$ are n shared fully connected layers, $LE_1, \dots, LE_m$ are m separate layers for explanation units $e_1, e_2, \dots, e_K$, and $LL_1, \dots, LL_k$ are k separate layers for class labels $c_1, \dots, c_L$	120
8.2	Impact of α . The accuracy and reliability results with respect to varying α in $[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^0, 10^1, 10^2, 10^3, 10^4]$	129

List of Tables

2.1	Example dataset	11
3.1	Summary of explanation methods	38
4.1	Synthetic transaction data	40
4.2	Frequent pattern set with threshold $minSupp = 0.6$	41
4.3	Synthetic vector/tuple data	41
4.4	Corresponding transactional data	41
4.5	Synthetic sequence data	42
4.6	Example dataset $\mathcal{T}$ for contrast patterns	43
4.7	Dataset used to illustrate the equivalence class	50
4.8	Closed pattern and generator	50
4.9	The contingency table for pattern p in dataset $\mathcal{T}$	54
4.10	Dataset in original feature space	56
4.11	Dataset in pattern space	56
4.12	Connection between frequent pattern mining and subspace search	59
5.1	Comparison of local explanation methods.	64
5.2	Test instances to illustrate explanations.	70
5.3	Precision and recall results for synthetic data.	74
5.4	F_1 (faithfulness) results for synthetic data.	74
5.5	UCI datasets description.	74
5.6	Precision and recall results of PALEX, aLIME, LIME and Global on UCI datasets (the best results are highlighted in bold, and standard deviations are included in parentheses)	76
5.7	F_1 (faithfulness) results of PALEX, aLIME, LIME and Global on UCI datasets (the best results are highlighted in bold, and standard deviations are included in parentheses)	77
5.8	Faithfulness (F_1 score) of different sampling methods on numeric data	77
5.9	Impact of selection process. #Expl denotes the number of explanations, “before” denotes the metrics before the pruning process and “after” denotes the metrics after the pruning process.	79
6.1	Explanation quality on synthetic datasets (the best value in bold; ‘-’ means method does not apply on corresponding dataset, e.g., Synthetic-1 is for numeric explanation and binary explanation method does not apply).	95
6.2	Explanation quality on UCI datasets (the best value is highlighted in bold).	96
6.3	Explanations extracted with Random(N) and LEAP for the comment “The projects After Effects, Music, Foto, Web sites and another you can find and buy here http...” which is classified as Spam	100
7.1	Comparison of xPWL with traditional classifiers	102
7.2	Data description	110

7.3	Accuracy results where expert explanations come from decision trees (with the best values in bold). xPWL - our proposed method; PWL - Partition-wise linear models; LR - Logistic regression; LR-PCA - LR with PCA pre-processing; MLP - Multilayer perception; RF - Random forest. “*” denotes that the corresponding method is trained with expert explanations	111
7.4	Reliability results where expert explanations come from decision trees (with the best values in bold). xPWL - our proposed method; PWL - Partition-wise linear models; LR - Logistic regression; LR-PCA - LR with PCA pre-processing; MLP - Multilayer perception; RF - Random forest. “*” denotes that the corresponding method is trained with expert explanations	114
7.5	Accuracy results where expert explanations come from non-noisy features (with the best values in bold). xPWL - our proposed method; PWL - Partition-wise linear models; LR - Logistic regression; LR-PCA - LR with PCA pre-processing; MLP - Multilayer perception; RF - Random forest. “*” denotes that the corresponding method is trained with expert explanations	115
7.6	Reliability results where expert explanations come from non-noisy features (with the best values in bold). xPWL - our proposed method; PWL - Partition-wise linear models; LR - Logistic regression; LR-PCA - LR with PCA pre-processing; MLP - Multilayer perception; RF - Random forest. “*” denotes that the corresponding method is trained with expert explanations	116
8.1	Four scenarios on how reliability and accuracy of a prediction could influence a decision making process.	118
8.2	A toy dataset about a person blowing a balloon.	118
8.3	UCI datasets description.	123
8.4	Input required for the proposed method and the baselines. “Y” represents “Required” and “N” represents “Not required”. ExplNet - our proposed network where the explanation units are from the oracle; ExplNet(CP) - our proposed network where the explanation units are from contrast patterns; NN - Neural network without explanation subnetwork; NB - Naive Bayes; AdaBoost - AdaBoost; SVM - Support vector machine with RBF kernel; RF - Random forest. “*” denotes models trained with expert explanations in a multi-label setting.	126
8.5	Reliability results where expert explanations are extracted from decision trees (with the best values in bold). ExplNet - our proposed network where the explanation units are from the oracle; ExplNet(CP) - our proposed network where the explanation units are from contrast patterns; NN - Neural network without explanation subnetwork; NB - Naive Bayes; AdaBoost - AdaBoost; SVM - Support vector machine with RBF kernel; RF - Random forest. “*” denotes models trained with expert explanations in a multi-label setting.	127

8.6	Accuracy results with expert explanations are extracted from decision trees (with the best values in bold). ExplNet - our proposed network where the explanation units are from the oracle; ExplNet(CP) - our proposed network where the explanation units are from contrast patterns; NN - Neural network without explanation subnetwork; NB - Naive Bayes; AdaBoost - AdaBoost; SVM - Support vector machine with RBF kernel; RF - Random forest. “*” denotes models trained with expert explanations in a multi-label setting.	130
8.7	Reliability results where expert explanations come from decision trees (with the best values in bold). ExplNet - our proposed network where the explanation units are from the oracle; ExplNet(CP) - our proposed network where the explanation units are from contrast patterns; NN - Neural network without explanation subnetwork; NB - Naive Bayes; AdaBoost - AdaBoost; SVM - Support vector machine with RBF kernel; RF - Random forest. “*” denotes models trained with expert explanations in a multi-label setting.	131
8.8	Accuracy results where expert explanations come from decision trees (with the best values in bold). ExplNet - our proposed network where the explanation units are from the oracle; ExplNet(CP) - our proposed network where the explanation units are from contrast patterns; NN - Neural network without explanation subnetwork; NB - Naive Bayes; AdaBoost - AdaBoost; SVM - Support vector machine with RBF kernel; RF - Random forest. “*” denotes models trained with expert explanations in a multi-label setting.	132

Chapter 1

Introduction

1.1 What is interpretability?

Recently, interpretability has been recognized as an important metric for machine learning models. Interpretability refers to the ability that users can understand that how a prediction of a model is derived. An interpretable model is a model whose structures/parameters can be understood by users to justify its predictions. E.g., decision trees, linear models and pattern based methods are known to be interpretable models. The prediction paths in decision trees, the weights/coefficients in linear models and the patterns in pattern based methods are all understandable by humans.

1.1.1 Why does interpretability matter?

Interpretability of machine learning models can be essential for their deployments, especially in domains like medical analysis, criminal analysis, and marketing where wrong predictions can result in severe consequences. This is due to the fact that most of traditional classifiers are solely trained to optimize loss functions in terms of the accuracy and there is no guarantee that the logical reasoning of their predictions is reasonable or meets experts' expectation. Knowing that a prediction is made based on reasonable evidence by the model, helps users to understand the prediction better. Thus they can accept or reject the prediction with greater confidence. Moreover, explanations for predictions are mandatory in some cases, i.e., the European Union's new General Data Protection Regulation, which took effect in 2018, grants users the right to ask for an explanation of any algorithmic decision made about them, such as the refusal of a loan application (Goodman and Flaxman, 2016).

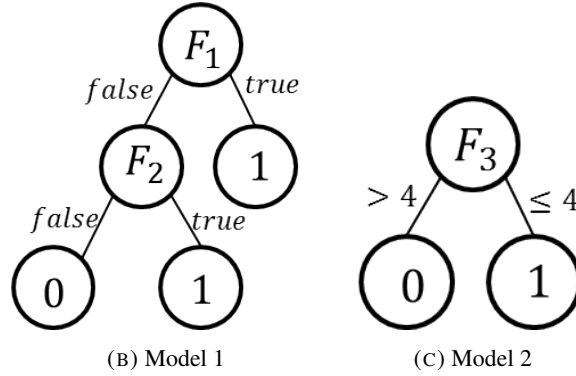
Users need to understand the predictions of a classifier, especially when decisions based on the predictions can have severe consequences. The explanation of a prediction reveals the reason why a classifier makes a certain prediction and it helps users to accept or reject the prediction with greater confidence. A prediction alone does not tell much about the model or data, and it is no better than random guess if it cannot be explained in a meaningful term. For example, in the medical diagnosis scenario, a patient cannot be convinced of any result if the explanation is "the machine says so".

Different classifiers may make the same prediction based on different evidence. Consider the example in Figure 1.1, where the dataset consists of four instances such that each instance is represented by three features (F_1 , F_2 , F_3) and a class. Assume the true underlying model is: $class = 1$ IF $F_1 = true$ OR $F_2 = true$. Two decision trees, model 1 and model 2, are trained from the data. Although both of them are able to achieve perfect accuracy, they make predictions based on different evidence. For instance x_1 , model 1 gives the prediction $class = 1$ because " $F_1 = true$ ", while model 2 gives the prediction $class = 1$ because " $F_3 \leq 4$ ". Since F_3 should not be the supporting evidence, one could easily tell the prediction made by model 2 is untrustworthy. Even though F_3 has predictive

power, it is still the “wrong evidence”. A similar example in a real-life problem can be found in (Kaufman et al., 2012), where patient ID is associated with the class label in the patient data, but its predictive power is unexpected.

	F_1	F_2	F_3	$class$
x_1	<i>true</i>	<i>false</i>	1	1
x_2	<i>true</i>	<i>true</i>	2	1
x_3	<i>false</i>	<i>false</i>	5	0
x_4	<i>false</i>	<i>true</i>	3	1

(A) Dataset



(B) Model 1

(C) Model 2

FIGURE 1.1: Example of two accurate models based on different evidence

1.1.2 Trade-off between interpretability and accuracy

Though the predictions made by interpretable models (e.g., decision trees and logistic regression models) can be easily explained from their structures or parameters, these models usually fail to achieve high accuracy. On the other hand, many state-of-the-art models (e.g., neural networks, random forest and support vector machine) are able to achieve higher accuracy, but they are difficult or even impossible to interpret due to their model complexity. The trade-off between interpretability and accuracy is shown in Figure 1.2 (note that the figure shows the conceptual trend between interpretability and accuracy, but does not reflect the relationship among these classifiers which may behave differently in different circumstances). In general, the higher accuracy a model can achieve, the lower interpretability the model has. High accurate models usually are non-linear, non-smooth while the high interpretable models contain well defined relationships.

1.2 Explainability

Explainability refers to the ability that a model can provide explanations to justify its predictions. The explanations reveal the logical reasoning behind a model and give users greater confidence to accept or reject the predictions.

1.2.1 Different between interpretability and Explainability

Interpretability concerns that the internal mechanism of a model can be interpreted in human language and can be understood by users. For example, a decision tree is represented as a set of paths from the root and each path is a prediction branch with the corresponding

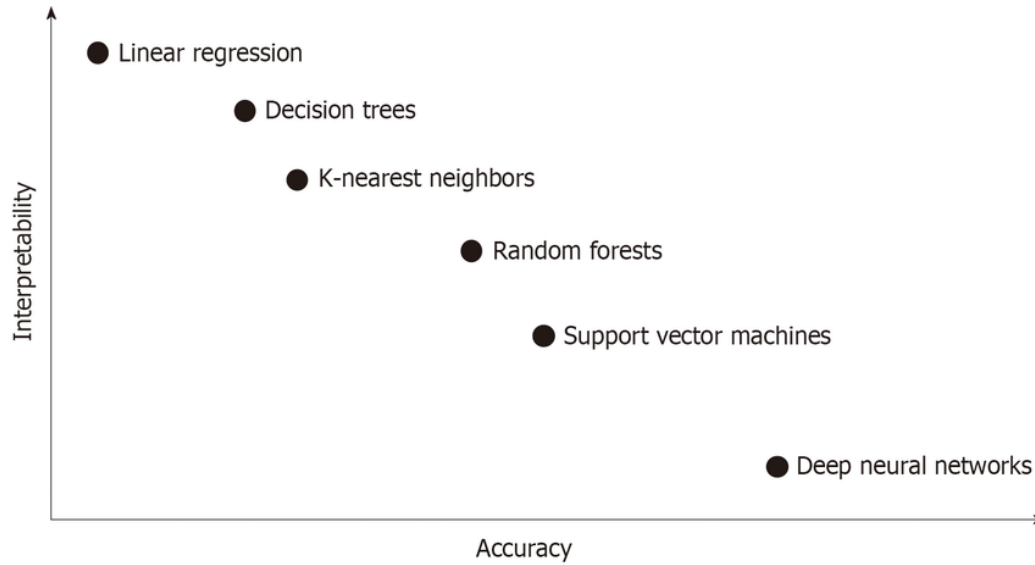


FIGURE 1.2: Trade-off between interpretability and accuracy

conditions. The internal structure of a decision tree can be easily interpreted and understood such that every prediction made by a decision tree corresponds to a path in the tree and the logical reasoning is obvious.

Explainability is an important aspect of interpretability. It does not require that the internal mechanism of a model be interpretable as long as the predictions of the model can be explained with explanations. For example, the explanation for the prediction made by a decision tree is the conditions in the prediction path.

Some models are interpretable and some models are not due to the complexity of the models. However, all models can be explainable. For the interpretable models, the explanations can be directly inferred from the internal mechanism of the models, and for non-interpretable models, the explanations can be extracted by model-agnostic explanation methods such as LIME (Ribeiro, Singh, and Guestrin, 2016c).

1.2.2 Types of explanations

The explanations can be categorized to types of explanations: **global explanations** and **local explanations**. Global explanations are the model-level explanations that provide insights for all predictions of a model, and local explanations are the instance-level explanations that provide insights for the individual predictions.

Given the decision tree shown in Figure 1.3, the global explanation for the model is that “features temperature, headache, cough, blood pressure, ... are used to build the tree”. Given an instance (temperature = 38.5, headache = yes, ...), the local explanation for this particular instance is that “features temperature and headache are used to reach its prediction by the tree”.

The research of this thesis focuses on local explanations. Unless specified, explanations refer to local explanations through this thesis.

1.2.3 How to get the explanations?

For interpretable models, the explanations can be directly inferred from the models’ structures/parameters. E.g., a prediction path in a decision tree indicates that what features and feature values are used by the tree to reach the prediction; weights/coefficients in a linear model suggests the feature importance especially in the case where all feature values

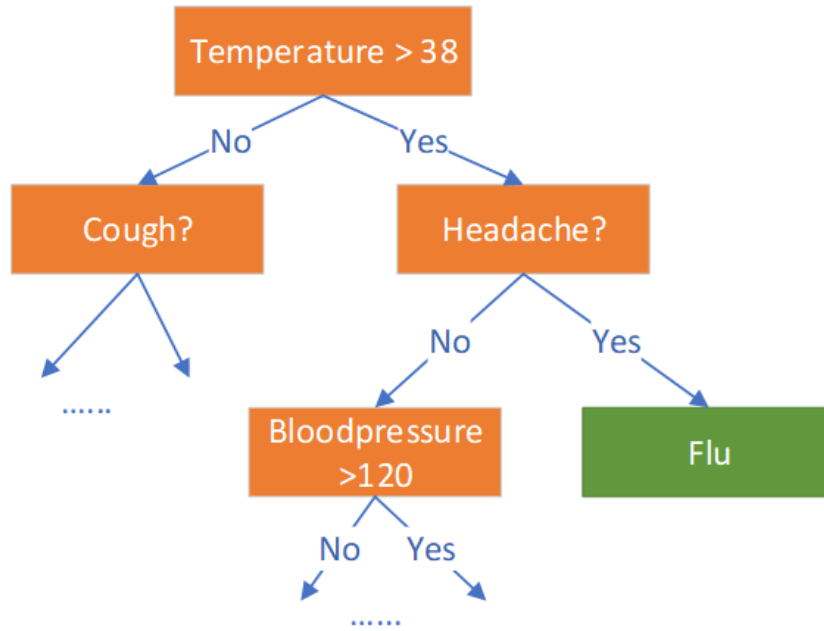


FIGURE 1.3: Example of a decision tree

are in the same scale which can be done via normalization; patterns in pattern based (or called rule based) methods are intrinsically understood by users. We also refer these interpretable models as white-box models.

On the other hand, for complex models like neural networks, random forest and support vector machine, it is difficult to directly extract explanations from the models' structure/parameters. These models need model-agnostic explanation methods to help extract explanations for them.

1.2.4 Improving performance of models using explanations

Explanations are also the interface between the model and the users, and they can be used to monitor the models. If the explanations do not look right, the users may know what the problems are and further resolve the issues. For example, explanations can help user to identify unexpected features that are used for training, e.g., if a medical diagnosis model keeps making decisions using the patient ID information, it can be retrieved that a data-preprocessing procedure is required to remove the patient ID feature.

Besides, the explanations can be used to supervise the learning process as well. The training data distribution may not be the close to the underlying true distribution, and it is difficult to collect more training data because it is expensive or even the new data can be more noisy. Our experiments suggest that by supervising with small amount of explanations from expert knowledge, the performance of the models can be improved, as the expert explanations may be able to cover the information that cannot be collected in the data collection process.

1.2.5 Properties of explanations

We investigate two properties of explanations: **faithfulness** and **reliability**. Faithfulness measures to which extent an explanation reflects the true logical reasoning of a model. Reliability measures to which extent an explanation matches the domain information or expert knowledge.

Take the tree shown in Figure 1.3 as example. For a given instance (temperature = 38.5, headache = yes, ...) and the tree model, assume a model-agnostic explanation method returns an explanation saying that “feature headache and cough are used by the model”. Suppose the tree structure is known and it can be inferred that true model (local) explanation is that “features temperature and headache are used to reach its prediction by the tree”, then the generated explanation is not faithful as it disagrees with the true model explanation. Suppose the model structure is unknown, but a doctor sees the patient information and make a diagnosis saying that “the disease is flu because of the temperature and cough”, then the generated explanation is called not reliable, because it does not match the expert knowledge.

When explanations are generated from white-box models, the faithfulness is not an issue as the explanations are inferred directly from the model. However, when the explanations are generated via model-agnostic explanation methods, being faithful is essential as the explanations tell nothing about the predictions if they are not faithful.

Only if an explanation is faithful, can users decide whether it is also reliable or not. E.g., users may choose to accept/trust a prediction if its explanation is reliable, or reject it otherwise.

Other properties of explanations have been discussed in (Molnar, 2019):

- **Consistency:** it concerns that the explanations from different models that are trained on the same task or used to predict the same instance are similar. The explanations (e.g., the feature importance) generated from different models are consistent if they are very similar. For example, a SVM and a linear regression model are trained using the same training data and both produce similar predictions, if both explanations from these two model for the prediction of an instance agree with each other, these explanations are consistent. This property may not be desired if there exist multiple true explanations, and explanations from different models can all be reasonable though they may not be similar.
- **Stability:** it concerns that the explanations of similar instances from the same model are similar. While consistency compares explanations from different models, stability compares explanations of similar instances from a fixed model. For example, given two instances that differ in only one feature and the difference is marginally small, if the explanations of their predictions by a model are similar, the explanations are stable. High stability suggests that slight variations in the features of an instance do not substantially change the explanation (unless these slight variations also strongly change the prediction). On the other hand, low stability suggests high variance of the explanation method. High stability is always desirable.
- **Comprehensibility:** it concerns how well humans can understand the explanations. This property is mainly associated with the representations of explanations and it depends on the audience. Measuring methods include measuring the size of an explanation (e.g., number of features in a vector explanation and number of conditions in a pattern explanation) or testing how well humans can predict the behavior of a machine learning model from its explanations. A complex transformation of features could be less comprehensible than the original features, e.g., when kernel methods are applied, the original features are projected into a new feature space using the kernel functions and the new features might be difficult to interpret.
- **Certainty:** it requires that an explanation contains the information about the certainty of the predictions by a machine learning model. Many machine learning models only return predictions without a indication about the models confidence

that the prediction is correct. An explanation that contains the certainty can be a complimentary to a prediction.

1.2.6 Decision making process with explanations

The comparison of decision making process with and without explanations is shown in Figure 1.4. In the traditional decision making process (Figure 1.4a), after a model is trained on the training data, a query is passed to the model to reach a prediction, then the prediction is used for decision making. In this case, accuracy of the model is the main factor that is considered. In the scenario (Figure 1.4b) where the explanations are involved, after a model reaches a prediction, the model and the prediction are passed to an explanation system to generate an explanation for the prediction, then both the prediction and the explanation are considered to reach a decision.

When the explanations are involved, they help to provide more information about the predictions and give users higher confidence to make use of the prediction, thus potentially facilitate the successful deployment of the model.

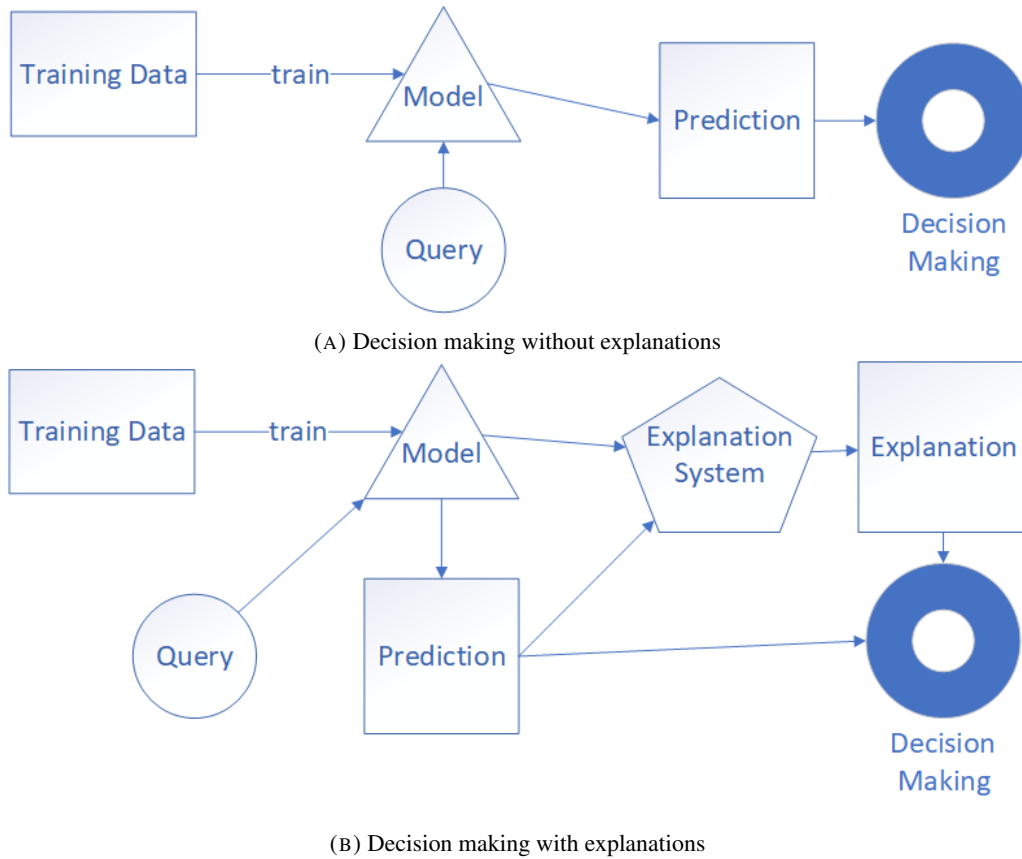


FIGURE 1.4: Demonstration of decision making process

1.3 Relationship of physical world, machine learning model and human

Machine learning models are trained on data collected from the physical world and used to provide valuable information such as predictions or visualizations. We investigate the three main entities that are involved: the physical world, the model and the human:

- Physical world refers to the real world where humans live and large quantity of data is being generated.
- Machine learning model refers to the algorithmic procedure that takes inputs and returns outputs with machine learning techniques.
- Human can be any user who manipulates the models or handles the model outputs.

The relationship of physical world, model and human are illustrated in Figure 1.5. **Data** is the connection from reality to model such that the data is collected from the physical world and the model is trained on the data. **Prediction** is the connection from machine learning model to human such that the prediction is the output of the model and human retrieves information via the prediction. **Knowledge** is the connection from physical world to human such that human gains knowledge (e.g., common senses and physical laws) from the reality.

However, there are always gaps among them:

- The gap between physical world and model comes from the data collection process which is practically impossible to make the training data distribution the same as the underlying true distribution if there exists one.
- The gap between physical world and human is due to limitation of human's perception about the physical world and is also inevitable.
- The gap between model and human is due to the lack of understanding about the models that make the decision such that users do not know how a prediction is made, when to trust the model, or how to fix errors.

Though the gaps cannot be erased, they can be mitigated. One connection between any two entities can be used to mitigate the gap of other connections. For example, the expert knowledge can help to mitigate the gap between physical world and model by methods like data pre-processing and feature engineering.

How can explainability help? The prediction alone does not tell much about the data, and it is useless to mitigate the gaps between reality and model, or between reality and human. Only when the prediction of a model can be explained, can human gain valuable insights from it to reach new knowledge, or can the flaws in data collection can be detected and corrected.

1.4 A possible solution to improve explainability - patterns

Pattern based methods have been shown to have discriminative ability. Due to its nature of being interpretable, we utilize patterns to improve the explainability of classification models.

Patterns are conjunctions of feature-value conditions, and they have been widely adopted in the interpretability research community as they are intrinsically easy to interpret. Given the data shown in Figure 1.1a, an example of pattern can be $(F_1 = \text{true}, F_2 = \text{false})$ or $(F_3 > 4)$. In particular, every prediction path in a decision tree can be reformed as a pattern.

We are interested in two types of patterns: **frequent patterns** and **contrast patterns**. Frequent patterns are the patterns that occur frequently in a dataset. E.g., assume a dataset consists of 100 instances and there is a user-specified threshold (called min support) which is 0.2, then a pattern that matches more than 20 ($100 \cdot 0.2$) instances is called a frequent pattern. Contrast patterns are the frequent patterns that occur more frequently in a dataset

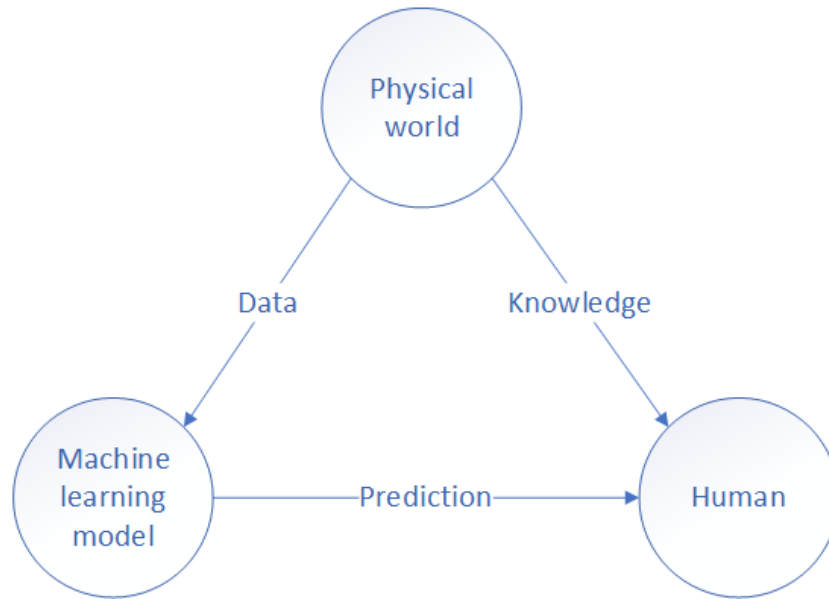


FIGURE 1.5: Relationship of reality, model and human

than another. E.g., assume a frequent pattern occurs 10 times in a dataset and occurs 2 times in another dataset, and there is a user-specified threshold (called min growth ratio) which is 5, then this pattern is a contrast pattern. Formal definitions of patterns are given and discussed in Chapter 2.

1.4.1 How can patterns improve explainability of models?

Patterns can be used to improve the model explainability as follows.

- Using patterns as features. When applying non-linear dimension transformation methods like principal component analysis (PCA), new features might not be interpretable, and when applying linear dimension transformation methods like linear discriminant analysis (LDA), the discriminative ability becomes an issue. Pattern based feature engineering utilizes pattern's discriminative power and its interpretability such that the pattern features can be both discriminative and interpretable.
- Using patterns as the representation of explanations. In this way, the problem of generating explanations can be transformed to pattern mining problem.
- Using patterns to build interpretable models. A common way is to build ensemble models with patterns. In particular, patterns can be used in partition-wise linear models (Oiwa and Fujimaki, 2014), such that each partition is represented by a pattern and associated with a local linear model. The pattern based partition-wise linear models preserve the interpretability and improve the accuracy by patterns' predictive ability.

Details of using patterns to improve the explainability will be discussed in the following chapters.

1.4.2 Patterns as explanations versus other forms of explanations

Explanations also can be found in other forms such as vectors of weights and instances. Vectors of weights use one number to represent the contribution of a feature, and instances

are the examples that are similar to the query instances. Explanations in the form of patterns have the following advantages:

- Patterns can be easily interpreted. The meanings of numeric values of vectors of weights can be confusing, e.g., a feature with contribution of 0.4 is not necessarily two times more important than a feature with contribution of 0.2 given different feature domains. Moreover, vectors of weights and instances are difficult to explain for large dimensional data.
- Patterns can be more precise. For example, patterns may contain the information about the value range of a feature, which cannot be indicated in other forms.
- Patterns can be found in complex data. For example, patterns are presented as subgraphs in the graph data, and subtrees in the tree data.

1.5 Thesis structure

The thesis is organized as follows:

- The background information and preliminaries that are used in the thesis are introduced in Chapter 2.
- The literature reviews about the trending interpretable models and model-agnostic explanations methods are presented in Chapter 3.
- Chapter 4 discusses the pattern based interpretable feature engineering methods. The contribution of this chapter is to summarize the framework of pattern based interpretable feature engineering and discuss different forms of mapping functions.
- Chapter 5 presents the proposed method that helps provide explanations for any classifier using contrast patterns. The contribution of this chapter is to propose a pattern based model-agnostic explanation method that achieves higher faithfulness than the baselines, and is able to generate multiple explanations when required.
- Chapter 6 investigate the relationship of local intrinsic dimensionality and the quality of explanations, and proposes a novel method to generate the synthetic neighborhood of a query instance, which is a crucial process for local model-agnostic explanation methods. The contribution of this chapter is that the proposed method improves the quality of local explanation methods and it can work with any existing local explanation method that involves local synthetic neighborhood generation.
- Chapter 7 proposes a pattern based interpretable classifier that aims to increase the accuracy of white-box classifier by enriching the hypothesis set without increasing the model complexity. The contribution of this work is propose an interpretable model that is able to achieve comparable accuracy as the black-box classifiers.
- In Chapter 8, we utilize the framework of multi-task learning in neural networks to build a model that is supervised by the class labels and expert explanations. The contribution of this work is to propose a method that is trained to make both accuracy and reliable predictions while the traditional classifiers are solely trained in terms of accuracy.
- Chapter 9 gives the conclusion and discusses the future work.

Chapter 2

Background

2.1 Introduction

In this chapter, we introduce the preliminaries that are used through this thesis. We start with the basic concepts of machine learning. Then we give the definitions related to patterns. Finally, the explanation terminologies and evaluation methods are discussed.

2.2 Basic concepts

2.2.1 Classification and pattern

Instance and dataset

An *instance* is a set of attribute-value pairs. The domain of an attribute can be discrete (where the attribute is called nominal/categorical) or continuous (where the attribute is called numeric/continuous). One nominal attribute is selected as the *class/label* of an instance. A *dataset* consists of a set of instances, where all instances have the same attributes with possibly different values. Such data is also known as tuple/vector data.

Example. An example dataset is shown in Table 2.1. This dataset consists of three feature F_1, F_2, F_3 and four instances x_1, x_2, x_3, x_4 . F_1 and F_2 are nominal features (especially they are also called binary features), and F_3 is numeric feature. The class label has two possible labels: 0 and 1.

	F_1	F_2	F_3	class
x_1	<i>true</i>	<i>false</i>	1	1
x_2	<i>true</i>	<i>true</i>	2	1
x_3	<i>false</i>	<i>false</i>	5	0
x_4	<i>false</i>	<i>true</i>	3	1

TABLE 2.1: Example dataset

Classifier

A *classifier* is a mapping relation $f(x)$ that maps an instance x (without class information) to a class label. Some classifiers are able to provide a confidence (or probability score) for the predictions.

Example. Given the dataset shown in Table 2.1, assume a classifier f is trained on the dataset, the input format of the classifier is the same as the format of an instance in the

dataset. For example, x_1 without the class label is a valid input for the classifier, and if the output of $f(x_1)$ is the same as the pre-defined class label of x_1 then the prediction is correct.

Pattern

A *pattern* is a conjunction of conditions. A condition may have one of three forms: i) $A = a$ or ii) $B \leq b$, or iii) $C > c$, where A is a nominal attribute, and B and C are numeric attributes. Patterns in complex data like graph or sequence data are discussed in Chapter 4.

Matching dataset

An instance x *matches* a pattern p if all conditions in p are true for x . The matching dataset of a pattern p in a dataset D is the subset of D where all instances in the subset match p , and is formally defined as $mds(p, D) = \{x \in D | x \text{ matches } p\}$.

Support

The support of a pattern p in a dataset D is the ratio of the size of the matching dataset to the size of D , and is defined as $supp(p, D) = \frac{|mds(p, D)|}{|D|}$.

Growth ratio

The growth ratio of a pattern p in dataset D_1 to dataset D_2 is the ratio of the corresponding supports, and is defined as $GrRatio(p, D_1, D_2) = \frac{supp(p, D_1)}{supp(p, D_2)}$. In particular, $GrRatio(p, D_1, D_2) = \infty$ if $supp(p, D_2) = 0$ and such patterns are known as jumping emerging patterns (Fan and Ramamohanarao, 2006). D_1 and D_2 can be two datasets, or two subsets of a dataset (i.e., divided by class labels).

Frequent pattern

A pattern p is called a frequent pattern in dataset D if its support is greater than a user-specified threshold $minSupp$.

Contrast pattern

Given two datasets D_1 and D_2 , a frequent pattern p is called a *contrast pattern* when the growth ratio $GrRatio(p, D_1, D_2)$ is greater than a user-specified threshold $minRatio$, which means p occurs more frequently in D_1 . Note that other metrics (e.g., support difference) can be used to define the contrast pattern. Interested readers can refer to (Dong and Bailey, 2012) for a discussion and comparison of the possible metrics.

Example. Taking the dataset D in Table 2.1 as an example, consider two patterns $p = \{F1 = true, F3 \leq 3\}$, $q = \{F2 = false\}$. The matching datasets of p, q are $mds(p, D) = \{x_1, x_2\}$, $mds(q, D) = \{x_1, x_3\}$, and the supports of both are $supp(p, D) = supp(q, D) = \frac{2}{4} = 0.5$. Both are frequent patterns if $minSupp < 0.5$. Assume the dataset is split into $D_1 = \{x_1, x_2, x_4\}$ and $D_2 = \{x_3\}$ by the class label, and the task is to find contrast patterns for D_1 . Then the growth ratios of p, q are $GrRatio(p, D_1, D_2) = \frac{2/3}{0} = +\infty$, and $GrRatio(q, D_1, D_2) = \frac{1/3}{1/2} = \frac{2}{3}$. p is called a contrast pattern for D_1 and q is not, if $minRatio$ is set to be greater than $\frac{2}{3}$.

2.3 Explanations

2.3.1 Representations of explanations

There are three typical representations of explanations: vector explanations, pattern explanations and instance explanations.

Vector explanations.

Vector explanations are represented in the form of vectors of weights, where each entry represents the contribution/importance of a feature.

Given a d -dimensional instance $x = (x_1, x_2, \dots, x_d)$ and a model f , the explanation for the prediction $f(x)$ is a d -dimensional vector $e_{f(x)} = (e_1, e_2, \dots, e_d)$. If e_i is binary such that $e_i \in \{0, 1\}$, $e_{f(x)}$ is called a binary vector explanation, where e_i is 1 only when i th feature is important for the prediction of $f(x)$. If e_i is arbitrary numeric value such that $e_i \in \mathbb{R}$, $e_{f(x)}$ is called a numeric vector explanation, where e_i represents the importance of i th feature.

Example. Assume a dataset consists of three features a_1, a_2, a_3 , and given a query x and a model f the explanation e of the prediction $f(x)$ by f is $(0, 0, 1)$, then e is a binary vector explanation and it suggests that only feature a_3 is important for the prediction.

Assume a dataset consists of three features a_1, a_2, a_3 , and given a query x and a model f the explanation e of the prediction $f(x)$ by f is $(0.5, 0.1, 0.2)$, then e is a numeric vector explanation and it suggests that $a_1 > a_3 > a_2$ in terms of the feature importance.

Pattern explanations.

Pattern explanations are represented in the form of vectors of patterns, where each condition in the pattern explains the logical reasoning of a prediction.

Example. Assume a dataset consists of three features a_1, a_2, a_3 , and given a query x and a model f the explanation e of the prediction $f(x)$ by f is $(a_1 = 3 \cap a_2 < 5)$, then e is an example of pattern explanation. This pattern explanation can be converted to a binary vector explanation as $(1, 1, 0)$ which suggests that features a_1, a_2 are important, but such conversion may result in information loss.

Instance explanations.

Instance explanations are instances that come from the training data. A typical example is the K-nearest-neighbor (KNN) model, especially when $K=1$.

Example. Given an instance x and a KNN model f for $K=1$, assume the explanation is x^* such that x^* comes from the training data, it suggests that x^* is the closest instance to x and therefore they have the same class label $f(x) = f(x^*)$.

2.4 Evaluation of explanations

2.4.1 Metrics

Given a model f and a D -dimensional instance x , assume $e' = \{e'_1, \dots, e'_D\}$ is the true vector explanation, and $e = \{e_1, \dots, e_D\}$ is the vector explanation generated. The quality

of e is measured by its closeness to e' . Given two representations of explanations, we define the following metrics.

When e, e' are numeric explanations, where $e_i, e'_i \in \mathbb{R}$, the quality of e , denoted by $Q(e|e')$, is measured using the cosine similarity:

$$Q(e|e') = |\text{cosine_similarity}(e|e')| = \left| \frac{e \cdot e'}{|e| |e'|} \right| \quad (2.1)$$

where $e \cdot e'$ is the dot product of two explanations, and $|e|$ calculates the l^2 -norm of e .

The closer $Q(e|e')$ is to 1, the higher the quality of e , and if it is 1 then e and e' are parallel.

When e, e' are binary explanations, where $e_i, e'_i \in \{0, 1\}$, $Q(e|e')$ is measured by the F_1 score:

$$Q(e|e') = \frac{2 * \text{recall}(e|e') * \text{precision}(e|e')}{\text{recall}(e|e') + \text{precision}(e|e')} \quad (2.2)$$

where $\text{recall}(e|e') = \frac{\sum_{i=1}^D e_i * e'_i}{\sum_{i=1}^D e'_i}$, which indicates how many chosen features (with non-zero explanation values in e) are truly important (with non-zero explanation values in e'), and $\text{precision}(e|e') = \frac{\sum_{i=1}^D e_i * e'_i}{\sum_{i=1}^D e_i}$, which indicates how many truly important features are chosen in e . $Q(e|e')$ equals 1 if and only if e and e' are identical, and is less than one otherwise.

Given a training dataset T , a model f and a D -dimensional instance x , assume p' is the true pattern explanation and p is the pattern explanation generated, we define the quality of the pattern explanations using Jaccard similarity:

$$Q(p|p') = \frac{|\text{mds}(p, T) \cap \text{mds}(p', T)|}{|\text{mds}(p, T) \cup \text{mds}(p', T)|} \quad (2.3)$$

where $\text{mds}(p, T)$ is the matching dataset of pattern p in the training data T . This metric is 1 when to pattern explanations are identical.

2.4.2 Faithfulness and reliability

Faithfulness describes that the relationship of an explanation with the true **model explanation**, which means that the true explanations should be extracted from models. An explanation is faithful if it is similar to the corresponding model explanation.

Reliability describes that the relationship of an explanation with the true **expert explanation**, which means that the true explanations should be extracted from experts knowledge or domain information. An explanation is reliable if it matches the expert knowledge. An explanation should be faithful in order to evaluate the reliability.

Both faithfulness and reliability can be evaluated using the metrics described in Section 2.4.1 with different ground true explanations.

When evaluating the faithfulness, the true explanations are directly from the models. Faithfulness is generally used to evaluate the quality of model-agnostic explanation methods (Ribeiro, Singh, and Guestrin, 2016c), and a common way is to obtain the ground truth from interpretable models while treating them as black-box models for the explanation generation methods. It is worthy pointing out that it is impossible to exactly evaluate the faithfulness of the model-agnostic explanation methods as the ground truth cannot be obtained from the black-box models that need the explanation methods.

When evaluating the reliability, the true explanations are obtained from experts using the domain knowledge (e.g., a doctor knows the symptoms to reach the diagnosis of a

patient, or a marketing analyzer knows the characteristics that attract customers). The reliability is used to evaluate the quality of the predictions of a model, and it differs from accuracy in the sense that a model could make a correct prediction but its logical reasoning is totally false. For example, patient ID information is discriminative when the patients are categorized by their diseases to arrange the ID, if such information is given to a model, it may become dominant due to its high correlation with the class label, but explanations using patient ID are not reliable.

Example. Given the decision tree given in Figure 1.3 and a patient's information (temperature = 38.5, headache = yes, ...), when providing the decision tree model to a model-agnostic explanation method (e.g., LIME (Ribeiro, Singh, and Guestrin, 2016c)) to seek an explanation for the prediction of the patient, if the explanation from the explanation method suggests that features (temperature, headache) are used for the model to reach the prediction, it is faithful as it agrees with the ground true model explanation (temperature > 38, headache = yes).

Given the decision tree given in Figure 1.3 and a patient's information (temperature = 38.5, headache = yes, ...), when providing the decision tree model to a model-agnostic explanation method (e.g., LIME (Ribeiro, Singh, and Guestrin, 2016c)) to seek an explanation for the prediction of the patient, if the explanation from the explanation method suggests that features (cough, bloodPressure) are used for the model to reach the prediction, it is not faithful as it disagrees with the ground true model explanation (temperature > 38, headache = yes).

Given the decision tree given in Figure 1.3 and a patient's information (temperature = 38.5, headache = yes, ...), the model gives the prediction "flu" and the explanation (temperature > 38, headache = yes). If a doctor knows the true symptoms are temperature and headache, then the prediction of the model is reliable as its explanation matches the domain knowledge.

Given the decision tree given in Figure 1.3 and a patient's information (temperature = 38.5, headache = yes, ...), the model gives the prediction "flu" and the explanation (temperature > 38, headache = yes). If a doctor knows the true symptoms should be cough and headache, then the prediction of the model is not reliable as its explanation does not match the domain knowledge.

Chapter 3

Literature Review

3.1 Introduction

In this chapter, we introduce the general interpretable models and the model-agnostic explanation methods that are related to our research. We also analyze the advantages and disadvantages of these methods. Some detailed discussions and closely related works of our proposed methods can be found in the corresponding chapters.

3.2 White-box models

White-box models are intrinsically interpretable and easy to use when the model interpretability is required. Common white-box models include linear regression, logistic regression, decision tree and pattern based methods.

3.2.1 Linear models

A linear regression model (Hastie et al., 2005) makes predictions using a weighted sum of the feature values, and it assumes linearity relationships that are easy to interpret.

Given an instance $x = (x_1, x_2, \dots, x_d)$, a linear model f gives the prediction as:

$$f(x) = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_d x_d \quad (3.1)$$

where α_0 is the intercept and $\alpha_i (i > 0)$ are the feature weights/coefficients.

More discussion and the variants of linear regression can be found at (Hastie et al., 2005).

The explanations for a linear regression model can be extracted from the weights/coefficients. The explanations are represented as feature importance (FI), and the feature importance of i th feature FI_i can be measured by the absolute values of its weight:

$$FI_i = |\alpha_i| \quad (3.2)$$

or by the absolute values of the its t-statistic (Molnar, 2019):

$$FI_i = \left| \frac{\alpha_i}{\sigma(\alpha_i)} \right| \quad (3.3)$$

where $\sigma(\alpha_i)$ is the standard deviation of the corresponding feature distribution. This metric increases with increasing feature weight and decreases with increasing feature variance (which means that we know less about the feature value). The sign of a weight indicates whether the feature has a positive contribution or a negative contribution.

Assume a linear model f is trained on a dataset that consist of three normalized features a, b, c , and is represented as:

$$f(x) = 5a - 3b + 4c - 2 \quad (3.4)$$

It can be interpreted as: feature a has the most impact on the predictions. A unit increase of feature a results in an increase of 5 for the outcome; A unit increase of feature b results in a reduce of 3 for the outcome; A unit increase of feature c results in an increase of 4 for the outcome;

Advantages

The advantages of linear models include:

- The linear models are simple to implement and use.
- How the predictions is derived is clear: they are based on the weighted sums of the feature values.
- It is guaranteed to find the optimal weights given the training data.

Disadvantages

The disadvantages of linear models include:

- This type of models is based on the assumption that the features and class labels have linear relationships. For non-linear data, kernel methods or other feature transformation methods can be applied, but doing this could increase the complexity thus reduce the interpretability.
- The linear models usually fail to achieve high accuracy comparing with the state-of-the-art models like random forest and neural networks.
- The explanation as a vector of weights can be misleading. A feature that is positively correlated with the class label may have negative weight and vice versa when there are other features that are highly correlated with it.
- The features are required to be normalized or in the same scale in order for the weights to be meaningful.

3.2.2 Logistic regression

While the linear regression works well for regression task, it fails to perform on classification problems due to the lack of the ability to output classification probability. The solution for classification is logistic regression which utilize the logistic function:

$$\text{logistic}(\eta) = \frac{1}{1 + \exp(-\eta)} \quad (3.5)$$

Given an instance $x = (x_1, x_2, \dots, x_d)$, a logistic regression model gives the probability $p(y = C)$ of predicting a certain class label C , such that:

$$p(y = C) = \frac{1}{1 + \exp(-(\alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_d x_d))} \quad (3.6)$$

The explanation of the weights in a logistic regression model and the explanation of the weights in a linear regression model are different. The output in a logistic regression model is a probability within the range $[0, 1]$, and the weights do not affect the probability linearly. The weighted sum of feature values is converted by the logistic function to a probability score. One way to get an explanation for a prediction is to find out how the prediction output changes when a feature x_i is increased/decreased by 1 unit. This can be done by applying the exponential function to both sides of the equation:

$$\frac{P(y = C)}{1 - P(y = C)} = odds = \exp(\alpha_0 + \alpha_1 x_1 + \dots + \alpha_d x_d) \quad (3.7)$$

Then the change is usually calculated by the ratio of these two odds:

$$\frac{odds_{x_i+1}}{odds} = \frac{\exp(\alpha_0 + \alpha_1 x_1 + \dots + \alpha_i(x_i + 1) + \dots + \alpha_d x_d)}{\exp(\alpha_0 + \alpha_1 x_1 + \dots + \alpha_i x_i + \dots + \alpha_d x_d)} = \exp(\alpha_i) \quad (3.8)$$

The explanation of the prediction of an instance $x = (x_1, x_2, \dots, x_d)$ is a vector of weights $(\exp(\alpha_1), \exp(\alpha_2), \dots, \exp(\alpha_d))$ where each entry denotes that if there is a change in feature x_i by one unit then the odds ratio changes by a factor of $\exp(\alpha_i)$.

The advantages and disadvantages of linear model also apply to the logistic regression model. One advantage over the linear model is that the logistic regression model is able to give probabilities.

A disadvantage of logistic regression is that the explanation for a prediction is difficult to comprehend as the interpretation of the weights is multiplicative and not additive. Complete separation could be an issue for logistic regression. If the data can be perfectly predicted using only one feature, the logistic regression cannot be trained as the weight for that feature would be infinite.

As logistic regression is an interpretable model, it can be improved from the perspective of accuracy for different tasks while its interpretability is preserved (Lv and Chen, 2018) (Kocbek et al., 2019) (Jovanovic et al., 2016).

3.2.3 Decision tree

Linear regression and logistic regression models cannot handle the situations where the relationship between feature values and the class label is nonlinear. Tree based methods split the data according to the cutoff feature-value pairs that are calculated by different criteria. Each splitting creates several subsets of the dataset and the splitting process is repeated until all instances in the subset belong to the same class or the maximum tree depth is reached. The final subsets of the tree are called leaf nodes and the intermediate subsets are called split nodes. When predicting an instance, it starts from the root of a tree and reaches a leaf node along a prediction path on which the feature values of the instance satisfy all the conditions of the split node, and the average result (class labels for classification and numeric values for regression) is the final prediction.

There exist many algorithms to build a tree, and they mainly differ in the criteria that is used to find the splitting feature and values on a split node. The classification and regression trees (CART) algorithm proposed in (Breiman, 2017) decides the feature-value cutting pair by minimizing the variance of the prediction for regression tasks or the Gini index of the class distribution for classification tasks. The variance reveals that how much the predictions of the instances in a node are located around the mean value, and the Gini index evaluates the impurity of a node, e.g., if class labels of the instances in a node are equally distributed, the node is totally impure, and if all instances in a node belong to

the same class, the node is pure. At each splitting process, the node select the feature-value pair that results in the best partition with respect to the variance for regression or Gini index for classification. CART keeps searching and splitting until the criterion is satisfied.

The explanation of a prediction for a decision tree can be represented as a binary vector or a pattern. A binary vector explanation is of the same dimensionality as the instance, and each entry is 1 if the corresponding feature occurs in the prediction path. A pattern explanation consists of splitting features and their conditions.

There are works that extend the decision trees to improve the accuracy while preserving the interpretability in domains where the interpretability is required (Lakkaraju, Bach, and Leskovec, 2016) (Letham et al., 2015) (Wang and Rudin, 2015).

Advantages

The advantage of the decision tree models include:

- The interpretation of the prediction paths is arguably more intuitive and understandable.
- The tree structure can be visualized easily and rendered with the edges and nodes.
- The feature transformation is not need to learn a tree, while for the linear models, it is necessary to transform categorical features into numeric features.

Disadvantages

The disadvantages of the decision tree models includes:

- Decision tree models can not capture linear relationships.
- The trees suffer from lack of smoothness. A small change to the training data can result in a completely different tree; or if a different feature with the same variance or Gini index is chosen as the split feature of the root node, the final tree structure would be different as well.
- The trees cannot output confidence or probabilities for the predictions.
- Deep trees are not interpretable. The size of the explanations increases with the depth or length of the prediction path. A large size makes the explanations difficult to understand.

3.2.4 Naive Bayes Classifier

The Naive Bayes classifier proposed in (Maron, 1961) is constructed based on the Bayes' theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (3.9)$$

where $P(A|B)$ is the conditional probability or the likelihood of event A given that B is true, $P(B|A)$ is the similar defined as $P(A|B)$, $P(A)$ and $P(B)$ are the marginal probability of observing A and B independently.

The Naive Bayes classifier computes a class probability for each feature independently and the final probability of a class given all features is based on the individual class

probabilities. Formally, the Naive Bayes classifier computes the conditional probability of a class C as:

$$P(C|x) = \frac{1}{Z} P(C) \prod_{i=1}^n P(x_i|C) \quad (3.10)$$

where Z is the scale parameter that guarantees that the probabilities of all classes sum to 1.

The Naive Bayes classifier assumes that all features are independent, making it an interpretable model such that it is clear to see the contribution of each feature to the final prediction.

3.2.5 K-Nearest Neighbors

The k-nearest neighbor method (Altman, 1992) can be applied for classification tasks and regression tasks. It makes predictions based on the nearest neighbors of an instance. For classification tasks, the prediction of an instance is the majority class in the k nearest neighbors of the instance. For regression tasks, the prediction of an instance is the average prediction of the neighbors. The key procedure is to choose a proper value for k , and decide the distance function for any two instances.

The k-nearest neighbor method is different from the previous interpretable models because it is an instance-based learning algorithm.

The main advantage of k-nearest neighbor method is that it has no parameters to learn, thus there is no need to interpret the modular parameters of structures.

The main disadvantage comes from the choice of the distance functions. First of all, it is difficult to infer the underlying true distance functions from the data. Second, the differences of the distances among the instances become small for high dimensional data due to the curse of dimensionality.

Recently, many accurate interpretable models are built based on k-nearest neighbors (Ezghari et al., 2017) (Kroupa et al., 2019) (Fuchs, Gertheiss, and Tutz, 2015).

3.3 Model-agnostic explanation methods

Interpretable models usually achieve lower accuracy comparing with the state-of-the-art models like random forest, neural networks and support vector machine (SVM), which are also difficult to explain. The lack of interpretability brings a challenge for the deployments of these black-box classifiers. In this section, several model-agnostic explanation methods are introduced.

3.3.1 Partial Dependence Plot

The partial dependence plot (PDP) proposed in (Friedman, 2001) is used to visualize the interaction between the class label and the features. It is called “partial” as it is difficult to plot the interaction between the class and all features, and usually it can plot the relationship of the class and one or two features. PDP is helpful when a model is dominated by a few features.

Assume the input instance of a model f consists of n features, let z_S be a chosen subset of all features and z_C be the complement subset, the partial dependence function is defined as:

$$\bar{F}_{z_S}(z_S) = E_{z_C} [f(z_S, z_C)] = \int f(z_S, z_C) p_{z_C}(z_C) d(z_C) \quad (3.11)$$

where z_S are the chosen features for plotting the partial dependency (usually there are one or two features in z_S), and z_C are the remaining features that are also used by the

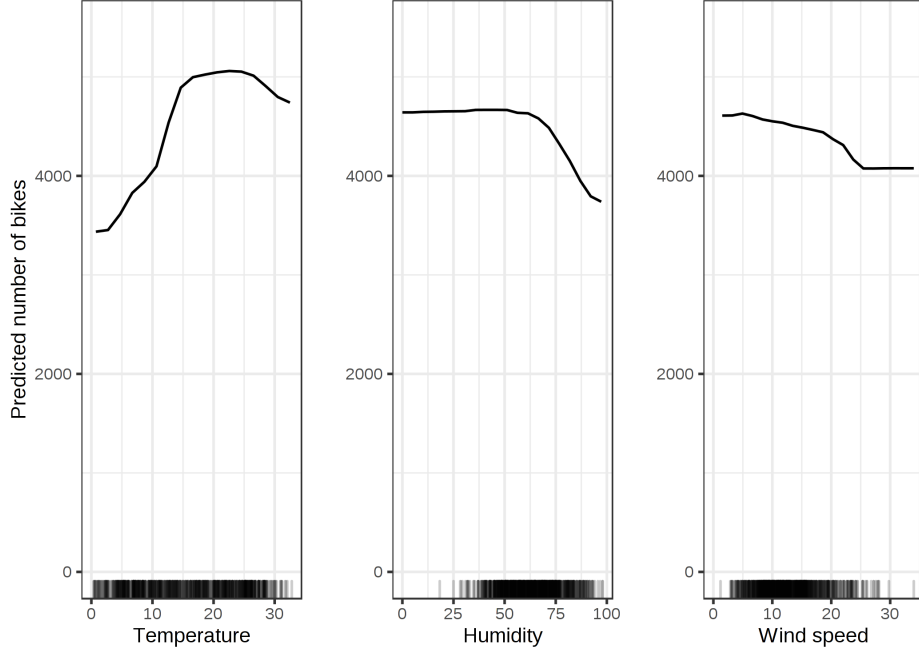


FIGURE 3.1: Partial dependency plots for the bicycle count prediction model and temperature, humidity and wind speed

model f . $p_{z_C}(z_C)$ is the marginal probability density of z_C which is formally:

$$p_{z_C}(z_C) = \int p(z_S, z_C) dz_S \quad (3.12)$$

where $p(z_S, z_C)$ is the joint density of all features, and the complement marginal density can be estimated from the training data, then Eq. 3.11 can be calculated by:

$$\bar{F}_{z_S}(z_S) = \frac{1}{n} \sum_{i=1}^N f(z_S, z_C^{(i)}) \quad (3.13)$$

where N is the number of instances in the dataset, $z_C^{(i)}$ are features values from i th instance.

The partial dependence plot method provide global explanations in the sense that it considers all instances in a dataset and analyzes the global relationship of a subset of features with the class labels.

An example is given in Figure 3.1 (Molnar, 2019). The task is to predict how many bikes will be rented on a given day. A random forest model is trained to analyze the partial dependencies of features temperature, humidity abd wind speed. The results are visualized in Figure 3.1. The first plot suggests that the number of rented bikes is high when the temperature is warm (roughly around 20 degree). The second plot shows that if the humidity is greater than 60%, the number of bikers reduces when the humidity keeps increasing. The last plot indicates that the higher the wind speed is, the less the probability that people go out for biking is. It is worthy noticing the that number of rented bikes does not keep reducing after the wind speed exceeds 25 km/h, and one possible reason is that the training data lacks instances where wind speed is higher than 25 km/h thus the model fails to make accurate prediction in this range.

Advantages

The advantages of partial dependence plot include:

- The definition of partial dependence is straight forward: the partial dependency of a particular feature is the average prediction if the values of the chosen features are applied to all instances and the rest features remain unchanged.
- If the chosen features are independent from other features, the PDP exactly reflects the effect of the chosen features on the predictions.
- The method is easy to compute and implement.

Disadvantages

The disadvantages of partial dependence plot include:

- It assumes that the chosen features are not correlated with other features, and this assumption usually does hold in reality. For example, when predicting a patient's diagnosis, it is not reasonable to assume that the blood pressure is not correlated with the weight.
- Partial dependency method can hide heterogeneous effects of features as it only computes the average effects. Assume a feature is positively correlated with the class label for half of all instances, and it is negatively correlated with the class for the other half, the partial dependency curve would be a flat line, as the effects of the feature are canceled by each half of the dataset.

3.3.2 Individual Conditional Expectation

Individual conditional expectation (ICE) (Goldstein et al., 2015) is a local version of the partial dependency plot, and it plots one line per instance to illustrate the effect for all instances when a feature changes.

ICE plots disaggregate the result of the partial dependency plot, which reflects the average partial effect of the chosen features, and it plots N estimated conditional expectation curves for N instances in a dataset. The PDP curve is the average of N ICE curves. Moreover, ICE can display the heterogeneous effects of features, which is a disadvantage of PDP.

A formal definition of the individual conditional expectation is: In an ICE plot, for each instance in $\{(z_S^{(i)}, z_C^{(i)})\}_{i=1}^N$ the curve $\bar{F}_{z_S}^{(i)}$ is plotted against $z_S^{(i)}$, while any other feature value $z_C^{(i)}$ is fixed, where N is the number of instances, z_S is the selected feature for visualization, and z_C is the complement feature set. Each curve of ICE represents the conditional relationship between z_S and the prediction at fixed values of z_C .

An example of ICE plot is shown in Figure 3.2 (Molnar, 2019). A random forest is trained for the prediction of the probability of cervical cancer for a woman given risk factors such as age. Each curve in the plot represents the probability across all ages for one woman. It can be seen that for most women the probability of getting cancer increases with increasing age. For some women with probability higher than 0.4, the age does not affect the probability significantly, which suggests that if a woman has a high probability at a young age, then the probability will not change significantly with age.

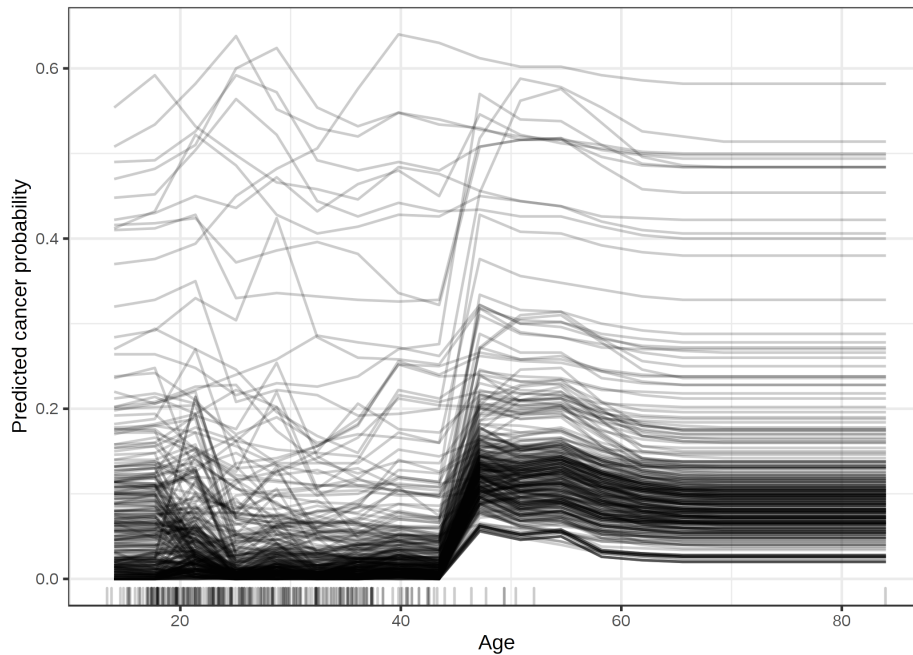


FIGURE 3.2: Individual conditional expectation plot of cervical cancer probability by age

Advantages

The advantages of ICE include:

- It is clear to see the effect of a chosen feature on a particular instance.
- Unlike the partial dependence plots, the individual conditional expectation plots can display heterogeneous relationships.

Disadvantages

The disadvantages of ICE include:

- ICE plot can only show the effect of one feature clearly, as two features would cause the drawing of overlaying surfaces.
- ICE assumes the same as PDP such that the chosen feature is independent from other features.
- With a large number of instances, the plot may become overcrowded.
- It is difficult to see the average influence from a ICE plot.

3.3.3 Feature Importance

The feature importance is defined as the increase of the prediction error (or the decrease of the prediction accuracy) of a model if the values of a selected feature is permuted to break the correlation between the class label and the chosen feature (Breiman, 2001) (Gregorutti, Michel, and Saint-Pierre, 2017). A feature is important if the prediction error increases by permuting its values, because the “absence” of the feature results in the loss of predictive power of the model. A feature is not important if the prediction error remains

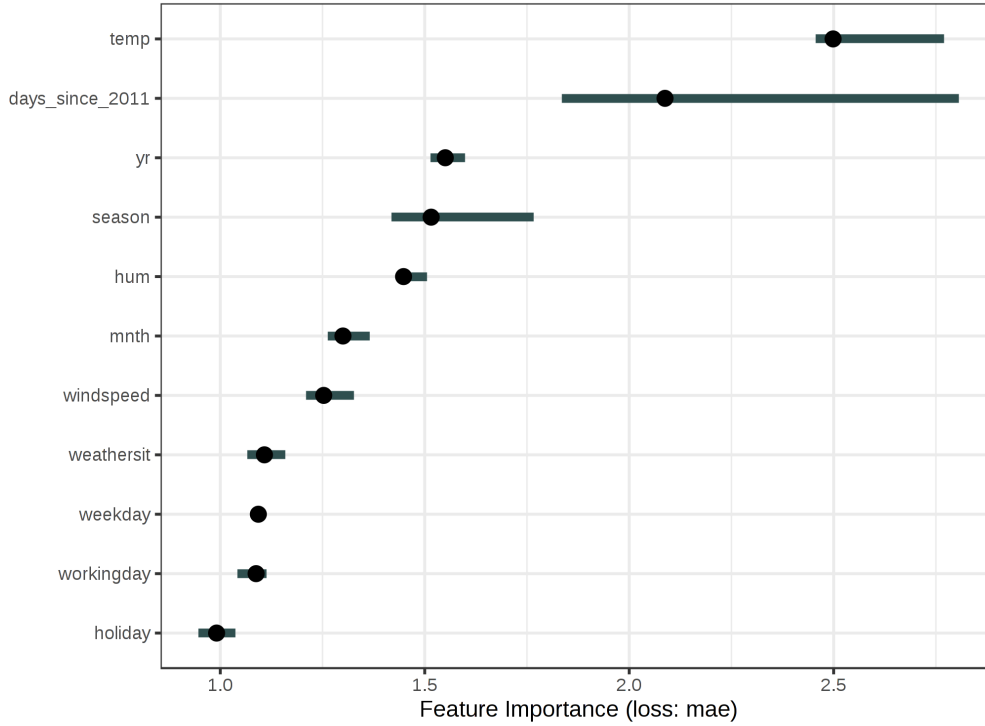


FIGURE 3.3: The importance for each of the features in predicting bike counts with a support vector machine.

unchanged when shuffling the feature values, in the sense that the selected feature can be ignored.

A general framework (Molnar, 2019) for computing the feature importance for any model can be described as:

1. Compute the model error e using loss functions such as mean absolute loss and mean squared loss.
2. For each feature i , do:
 - (a) Generate a synthetic dataset by permuting feature i in the original data.
 - (b) Compute the model error e' based on predictions of the synthetic instances.
 - (c) Calculate the feature importance of i by $FI_i = e'/e$ or by $FI_i = e' - e$
3. Return the feature importance FI in descending order.

An example is shown in Figure 3.3 (Molnar, 2019). A support vector machine model is trained for predicting the number of bikes that are rented, given the information of weather conditions and calendar. Mean absolute loss is used as the loss function. The figure suggests that temperature is the most important feature and holiday is the least important feature.

It is possible that one feature that is important to model M_1 may not be important to model M_2 . The research of Fisher, Rudin, and Dominici (Fisher, Rudin, and Dominici, 2018) defines model class reliance (MCR) as the range of feature important values across different models, taking into consideration that many prediction models possibly of different parametric forms can fit the data well.

Advantages

The advantages of feature importance include:

- The meaning of feature importance is clear such that people with little technique background can easily understand it.
- The feature importance methods implicitly take into account the correlations among the features. When permuting the values of a feature, the correlation of feature with other features are also destroyed.
- Many permutation based feature importance methods do not need the access to the internal structure of a model or the training data.

Disadvantages

The disadvantages of feature importance include:

- The feature importance depends on the error of a model. Different error measurements may result in different feature importance, and it is not clear to choose one over the others.
- The permuting process may generate different unrealistic synthetic instances that do not exist in reality, especially when the features are correlated. For example, it does not make sense to generate an instance saying "gender is male, pregnant status is yes".

3.3.4 Shapley Values

Shapley values can be explained with a simple scenario: when purchasing car insurance, a car owner needs to answer several questions regarding to the vehicle information, parking location and driver profiles, then the owner gets a quote. Assume the annual quote for the owner is \$1000 and the average quote for all cars is \$600. In this scenario, one may ask the question: how does each answer (e.g, car model, year, make, driver's age) influence the specific case comparing with the average prediction?

Shapley values can be adopted to answer this question. The calculation of Shapley values is based on all possible coalitions of feature values with and without a selected feature. However, exact calculation of Shapley values is computationally impossible due to the large amount of the permutations. Strumbelj et al. (Štrumbelj and Kononenko, 2014) propose an approach to estimate the Shapley value of j th feature with Monte-Carlo sampling:

$$\phi_j = \frac{1}{K} \sum_{i=1}^K \left(f(x_{+j}^i) - f(x_{-j}^i) \right) \quad (3.14)$$

where j is the index of the feature that the Shapley value is computed for, K is the number of iterations, f is the model being tested, and x_{+j}^i is the prediction of a modified x where a random number of feature values are replaced by the feature values of a random instance z_i , except for the j th feature. x_{-j}^i is almost the same as x_{+j}^i but the value of j th feature is also from the randomly picked instance z_i .

The Shapley value is different with feature importance in the sense that it does not measure the difference if a feature is "removed".

An example of the Shapley values is shown in Figure 3.4 (Molnar, 2019). A random forest is built for predicting the number of rented bikes for a particular day, given the information of weather and calendar. The average prediction for the rental bikes is 4518,

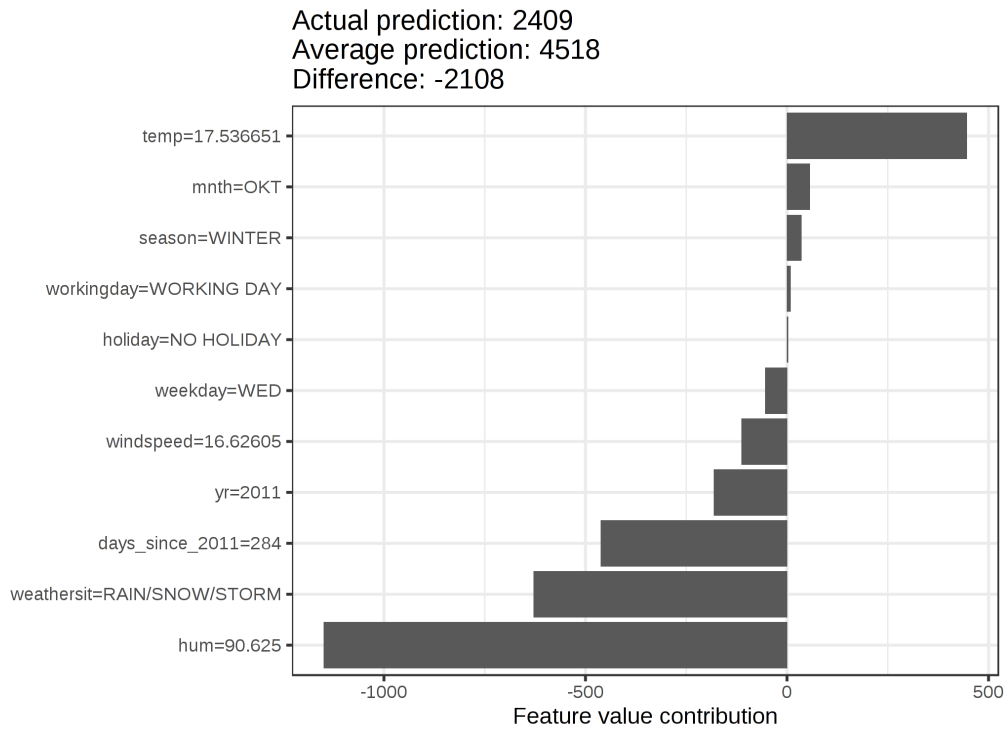


FIGURE 3.4: Shapley values for a certain day

and the prediction for the selected day is 2409, which is 2108 less than the average prediction. The Shapley values for all the features are displayed in the plot, from which it can be seen that the humidity is the main reason why the number of rental bikes is small, while the temperature still has a positive contribution to the prediction.

Advantages

The advantages of Shapley values include:

- Contrast explanation can be generated using the Shapley value by comparing an instance to a subset of instances or a single instance, instead of comparing the query to the entire dataset. This can be easily controlled by where the random instance is drawn.
- Shapley value has a solid theory, which gives the explanations a reasonable foundation.

Disadvantages

The disadvantages of Shapley values include:

- It is computationally expensive and in most cases only the approximate solution is feasible.
- The method need access to the training data.

3.3.5 Local interpretable model-agnostic explanations

Recently, a new framework local interpretable model-agnostic explanations (LIME) has been proposed in (Ribeiro, Singh, and Guestrin, 2016c) to generate explanations for individual predictions of black-box models. LIME extracts local explanations from local

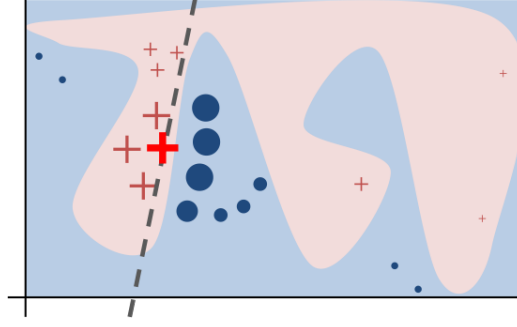


FIGURE 3.5: Example to present intuition for LIME

surrogate models (e.g., linear models) that are simple and interpretable. The local surrogates are built to approximate the behavior of a model in the vicinity of an instance.

Local surrogate models are interpretable models that are used to explain individual predictions of black box machine learning models. Local interpretable model-agnostic explanations (LIME) (Ribeiro, Singh, and Guestrin, 2016c) is a paper in which the authors propose a concrete implementation of local surrogate models. Surrogate models are trained to approximate the predictions of the underlying black box model. Instead of training a global surrogate model, LIME focuses on training local surrogate models to explain individual predictions.

The intuition of LIME is illustrated in Figure 3.5 (Ribeiro, Singh, and Guestrin, 2016c). The black-box model's underlying classification function f (that is unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the query instance that is being explained. LIME generates synthetic instances in the vicinity of the query instance, gets class labels (crosses or circles) using f , and weighs them by the proximity to the query instance (represented by size). The dashed line is the learned explanation that is locally (but not globally) faithful.

Mathematically, the framework of local interpretable model-agnostic explanations is defined as:

$$\text{explanation}(x) = \arg \min_{g \in \mathcal{G}} L(f, g, \pi_x) + \Omega(g) \quad (3.15)$$

where x is the query instance whose prediction is to be explained, π_x is a proximity measure between two instances to define the locality around x , $\mathcal{G}$ is a set of model candidates from where the local surrogate is drawn, and $\Omega(g)$ measures the complexity of the local surrogate (e.g., the depth for decision trees and the number of non-zero weights for linear models). The goal is to find a local surrogate g for an individual instance x and the explanation for the prediction of x is obtained from g .

The general procedure of LIME is described as following:

1. Select the instance to be explained.
2. Perturb the instance to get new synthetic instances in the vicinity of the query instance.
3. Weight the new synthetic instances according to their proximity to the instance.
4. Train a weighted, interpretable model on the synthetic dataset.
5. Explain the prediction by interpreting the local model.

An example of applying LIME on tabular data is shown in Figure 3.6 (Ribeiro, Singh, and Guestrin, 2016c). The task is to predict whether a mushroom is edible or not. Negative

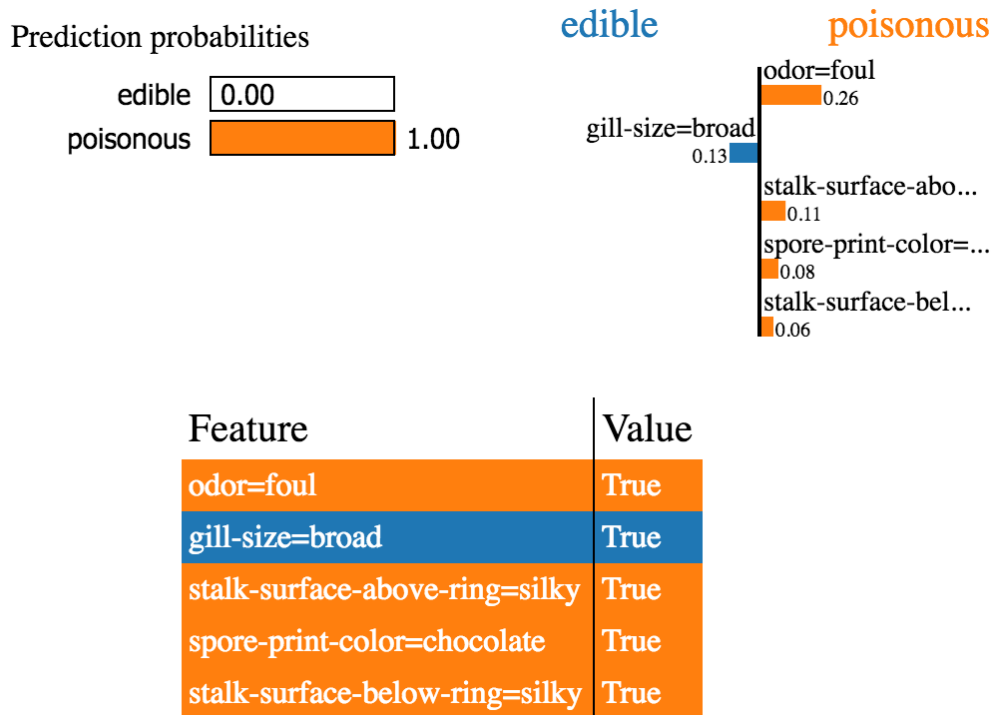


FIGURE 3.6: Example of LIME explanation on tabular data

(blue) features indicate edible, while positive (orange) features indicate poisonous. The feature odor contributes most to the prediction. It can be expected that if the feature odor is removed, the probability will reduce to $1.00 - 0.26 = 0.74$.

An example of applying LIME on tabular data is shown in Figure 3.7 (Ribeiro, Singh, and Guestrin, 2016c). The task is to predict whether the topic of an email is atheism or christian. Negative (blue) words indicate atheism, while positive (orange) words indicate christian. If the words “Host” and “NNTP” are removed from the document, it can be expected that the classifier predicts atheism with probability $0.58 - 0.14 - 0.11 = 0.31$.

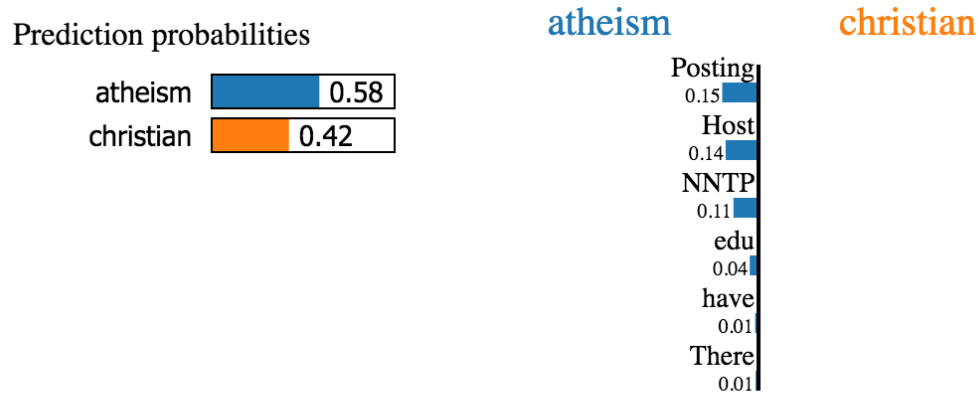
LORE (Local Rule-based Explanation) (Guidotti et al., 2018) follows the framework of LIME and adopts decision trees as local models. It first generates a set of synthetic neighbors through a genetic algorithm, and extracts a decision tree from the synthetic neighbors. A local explanation is derived from the tree structure.

Unlike LIME and LORE that generate synthetic neighbors of a given instance, LACE (Pastor and Baralis, 2019) selects K neighbors of the given instance from the training data and the extracts a set of classification rules. To generate local explanations, LACE requires access to the training data.

Advantages

The advantages of LIME include:

- LIME is able to work with different data types such as tabular data, text and images.
- LIME is flexible such that the local surrogate can be chosen from a variety of interpretable models.
- LIME is model-agnostic and does not need access to the internal structure of a model or the training data.



Text with highlighted words

From: johnchad@triton.unm.edu (jchadwic)
 Subject: Another request for Darwin Fish
 Organization: University of New Mexico, Albuquerque
 Lines: 11
 NNTP-Posting-Host: triton.unm.edu

Hello Gang,

There have been some notes recently asking where to obtain the DARWIN fish.

This is the same question I have and I have not seen an answer on the net. If anyone has a contact please post on the net or email me.

FIGURE 3.7: Example of LIME explanation on text data

Disadvantages

The disadvantages of LIME is described as follows:

- It is difficult to choose the proper distance measurement to define the locality.
- The process to generate the synthetic neighborhood of the query instance is critical to the quality of the explanations, but LIME simply samples the instances from a Gaussian distribution by random perturbations.
- The explanations are not stable. the explanations of two similar instances vary greatly in a simulated setting. Also, when the sampling process is repeated, the explanations that come out can be different.

3.4 Example-Based Explanations

Example-based explanation methods select representative instances of a dataset to explain the behavior of machine learning models or to illustrate the underlying data distribution. A intuitive example of these method is: if A is similar to B, and B is predicted as C, then it is reasonable to predict A as C.

3.4.1 K-nearest neighbors

The K-nearest neighbors model (see Section 3.2.5) is a typical interpretable machine learning model that provides examples as explanations. The selected neighbor instances are the explanations for a given query instance.

3.4.2 Adversarial examples

Adversarial examples are synthetic instances that are generated to fool machine learning models. These synthetic instances are results of small feature perturbations to true instances, and they cause a machine learning model to flip the predictions of the original instances. The adversarial examples are explanations in the sense that they provide insights of what costs to change the predictions and how vulnerable the models are.

For most adversarial example generation methods, the high level framework is to minimize the distance between an adversarial instance and the query instance such that the prediction of the adversarial example is shifted to the desired class.

The work proposed by Szegedy et. al (Szegedy et al., 2013) adopts a box-constrained optimization method to generate adversarial instances to fool machine learning models, and performs line-search to minimize the following function:

$$c \cdot |r| + \text{loss}(\hat{f}(x + r), l) \quad (3.16)$$

where x is the query instance (image or a vector of pixels is used in their proposed work), r is the change to x such that the new instance $x + r$ is classified as l by the given model f , $c \cdot |r|$ is the penalty function, and c is the penalty rate.

This method yields exact solution for convex loss functions and approximation solution for non-convex losses.

Goodfellow et. al (Goodfellow, Shlens, and Szegedy, 2014) proposed the fast gradient sign (FGS) method to generate adversarial instances for image data. FGS calculates the gradient of a model, and then computes the change based on the gradient such that the query instance. A step size ϵ is added or subtracted to the query instance/image according to the sign of the gradient for each feature/pixel. The idea is to change the feature values

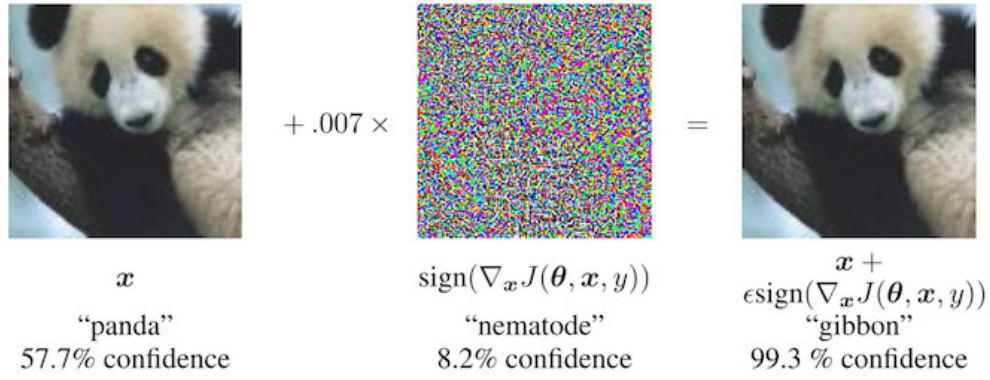


FIGURE 3.8: Making a panda look like a gibbon for a neural network

along the direction of gradient and intentionally alter the prediction. Formally, the fast gradient sign method is formulated as:

$$x' = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y)) \quad (3.17)$$

where $\nabla_x J$ is the gradient of the loss function of the model in terms of the input feature/pixel vector x , y is the true class label for x and θ is the model parameters (e.g., the weights of neural networks). The method only computes the sign of the gradient and does not need to calculate the exact value of the gradient. A positive sign represents that increasing the pixel value can increase the value of the loss function, and a negative sign represents that increasing the pixel value can decrease the loss.

An example of Goodfellow's work is shown in Figure 3.8. The original image (left image) that is classified as "panda" is then classified as "gibbon" (right image) by adding small perturbations (middle image) to it, while the new generated "gibbon" image still looks like a panda to humans.

While, the FGS method of Goodfellow needs to change many pixels in order to generate an adversarial example, the method proposed by Su et. al (Su, Vargas, and Sakurai, 2019) shows that it is possible to generate the image adversarial by modifying a single pixel.

The 1-pixel attack method aims to modify an image to be predicted as a desired class label by changing only one pixel. The method employs differential evolution (Storn and Price, 1997), which is a population based technique for complex multi-modal optimization, to find how to perform the manipulation (e.g., which pixel should be changed and what is the change value) on a query image. The perturbation is encoded into an array (candidate) and then is optimized using differential evolution. A candidate contains a fixed number of perturbations and each perturbation is a vector that contains five elements: x-y coordinates and RGB values. The initial number of candidates (population) is 400 and another 400 candidates (children) are added at each following iteration by using the differential evolution formula:

$$x_i(g+1) = x_{r1}(g) + F \cdot (x_{r2}(g) - x_{r3}(g)) \quad (3.18)$$

where each x_i is an element of the candidate solution, r_1, r_2, r_3 are random numbers, g is the index of current generation, and F is the scale parameter (which is set to 0.5). Each new child candidate solution represents a pixel that contains the information about the coordinates and RGB colors and each value is a mixture of three parent pixels that are randomly chosen. At each iteration, each candidate competes with the corresponding

parents according to the index of the population and only the winner survives for next iteration.

The work of Brown et. al (Brown et al., 2017) proposes adversarial patch that can be printed and added to any scene and presented to a image classifier such that the prediction of a query image is different with the prediction of the patched image. This method differs from other adversarial example generation in the sense that it does not require the new image to be close the query, and instead it place a patch on the certain location of an image.

While the methods that are discussed previously require the model structures to be known, the work by Papernot et. al (Papernot et al., 2017) proposes a black-box adversarial attack method and shows how to create adversarial examples without knowing the internal model structure or the training data.

Advantages

The advantages of adversarial examples include:

- They have good mathematical theory based on the models to be explained.
- There are many sophisticated optimization solvers that can be used to find adversarial examples.
- It works for most data types.

Disadvantages

The disadvantages of adversarial examples include:

- They are not “good” explanations due to their nature of being used to “attack” models instead of explaining models.
- There exist numerous adversarial examples given an instance, and it provides less meaningful or actionable information than the counterfactuals (see Section 3.4.3).
- The methods are computationally slow.

3.4.3 Counterfactual Explanations

Counterfactual explanations tell that how an instance can be modified to change its prediction. While most explanations serve the purpose to inform users how a prediction is reached, counterfactuals help to understand what could be changed to receive a desired prediction for a given instance. An example of a counterfactual explanation is similar to the following statement: “An applicant is rejected by a university because the math score is 60, if the math score is greater than 80, the applicant would be given an offer.”

Counterfactuals differ from adversarial examples mainly in the sense that counterfactuals aim to provide actionable and meaningful advice such as the changes to an instance are reasonable and understandable by humans, while adversarials are designed to attack models and the changes are significantly small.

The work in (Wachter, Mittelstadt, and Russell, 2017) generate counterfactuals by solving an optimization problem. Given any classifier f and the query instance x , the counterfactual x' for x is obtained by minimizing the following objective:

$$\min_{x'} \max_{\lambda} \lambda(f(x'), y') + d(x, x') \quad (3.19)$$

where $d(x, x')$ is a distance function that measures the distance of the counterfactual x' and the query instance x , and λ balances distance of features values and the distance of the predictions. The optimization can be solved by iteratively solving for x' for a fixed λ and then increasing λ until a desired solution is found.

An example can be found in (Wachter, Mittelstadt, and Russell, 2017) that shows counterfactual explanations for predicting the risk scores of diabetes for women of Pima heritage. The data consists of eight predictive features such as age, BMI, number of pregnancies and so on. A three-layer fully-connected neural network with two hidden layers of 20 neurons is trained to predict the risk scores, and L_1 or Manhattan distance is used as the distance function in Eq. 3.19.. The counterfactuals are generated to answer the question: what can be done for an individual to have a risk score of 0.5? The counterfactuals found are:

- Person 1: If the 2-hour serum insulin level was 154.3, the risk score would be 0.51.
- Person 2: If the 2-hour serum insulin level was 169.5, the risk score would be 0.51.
- Person 3: If the Plasma glucose concentration was 158.3 and the 2-hour serum insulin level was 160.5, the risk score would be 0.51.

The work in (Laugel et al., 2017) proposes an alternative way to discover the counterfactuals using the Growing Spheres algorithm. Instead of solving a time-consuming optimization problem, this method proposes a search algorithm. The method starts with a sphere around the query instance, and generates synthetic instances within the sphere. Then the synthetic instances are labeled by a given model. If none of the synthetic instances yields the desired prediction, the method contracts or expands the sphere until a counterfactual instance is found.

Advantages

The advantages of counterfactual explanations are:

- The counterfactual explanations are comprehensive, and can be easily understood by humans.
- The methods of generating counterfactual do not need to access the data or the models.
- The counterfactual methods can be applied to other non-machine-learning systems, as long as the systems take inputs and return outputs.

Disadvantages

The disadvantages of counterfactual explanations include:

- There are generally multiple counterfactual explanations and it is difficult to decide which one is the best.
- There is no guarantee that a counterfactual for a given query instance can always be found.

3.4.4 Prototypes and Criticisms

Prototypes are the a set of representative instances that are characterization of important aspect of a dataset. Case-based reasoning (CBR) (Slade, 1991) (Aamodt and Plaza, 1994) is a general paradigm for reasoning from past experience. It fits a new solution by retrieving and modifying previous solved problems, which can be treated as the prototypes.

However, CBR does not learn the underlying data distributions or structures. The work in (Kim, Rudin, and Shah, 2014) combines CBR and Bayesian generative framework and proposes Bayesian Case Model (BCM) to generate prototypes for classification. BCM finds prototypes by applying clustering algorithms that returns the center points as prototypes, and then uses the prototypes to infer the model parameters.

Using prototypes to summarize data relies on the assumption that the data distribution is well defined such that the prototypes are sufficient to represent the data, e.g., we only need two points (a center point and a border point) to represent a dataset if all points are uniformly distributed within a circle. However the assumption does hold in most cases. Thus it is important to provide insights about the part of input space where the prototypes do not provide good explanation. Another type of instances that do not fit a model is called criticism. Together with prototypes, criticisms help to better describe the data and model.

Kim et. al (Kim, Khanna, and Koyejo, 2016) proposes MMD-critic to discover both prototypes and criticisms based on maximum mean discrepancy (MMD). The maximum mean discrepancy (Gretton et al., 2007) measures the difference of two distributions P, Q , and it is defined as:

$$MMD(\mathcal{F}, P, Q) = \sup_{f \in \mathcal{F}} (E_{X \sim P}[f(X)] - E_{Z \sim Q}[f(Z)]) \quad (3.20)$$

When $\mathcal{F}$ is a reproducing kernel Hilbert space (RKHS) with a kernel function k , the supremum is obtained at:

$$f(x) = E_{X' \sim P}[k(x, X')] - E_{X' \sim Q}[k(x, X')] \quad (3.21)$$

The function f is called the witness function that maximum discrepancy between the two expectations in $\mathcal{F}$.

MMD-critic approximate the MMD using sample expectations by:

$$MMD^2(P, Q) = \frac{1}{n^2} \sum_{i,j=1}^n k(x_i, x_j) - \frac{2}{nm} \sum_{i,j=1}^{n,m} k(x_i, z_j) + \frac{1}{m^2} \sum_{i,j=1}^m k(z_i, z_j) \quad (3.22)$$

with the witness function:

$$f(x) = \frac{1}{n} \sum_{i=1}^n k(x, x_i) - \frac{1}{m} \sum_{j=1}^m k(x, z_j) \quad (3.23)$$

where k is a kernel function, n is the number of samples x_i from distribution P , and m is the number of samples z_i from distribution Q . MMD is 0 when two distributions are indistinguishable.

The process of finding prototypes is to minimize $MMD^2(X, X_S)$ where X is the dataset and $X_S \subset X$ is the prototypes. The criticisms are the instances that achieve the largest values of the witness function (Eq. 3.23).

MMD-critic can improve the interpretability of machine learning models by providing the prototypes and criticisms as explanations that improve users' understanding of the

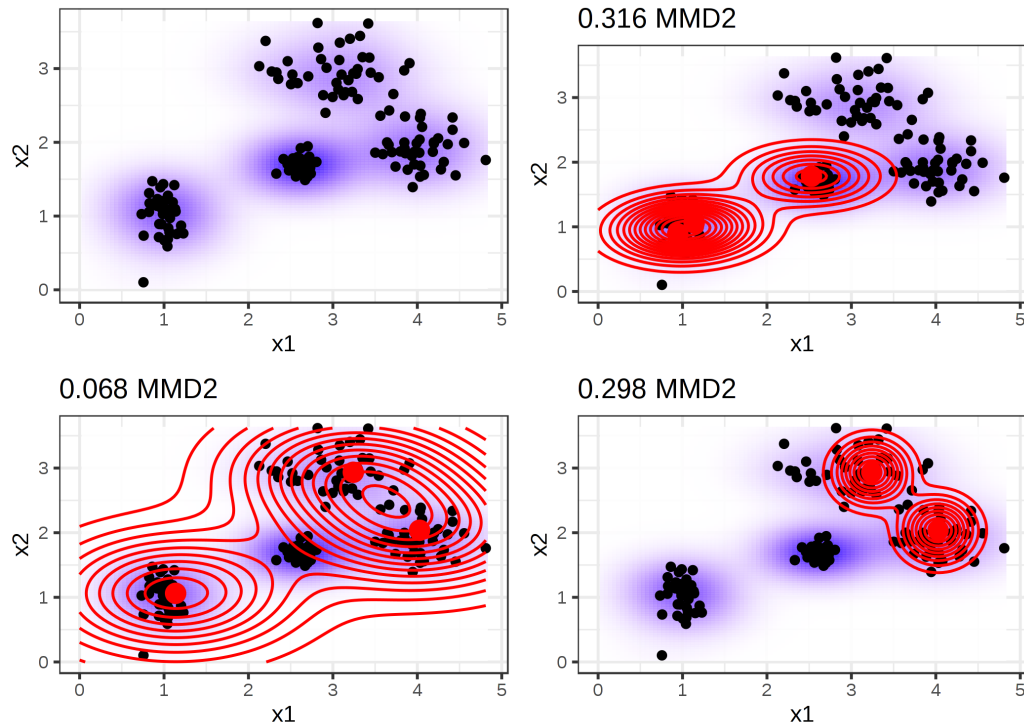


FIGURE 3.9: Demonstration of squared maximum mean discrepancy measure (MMD2) and different selections of prototypes.

data. Moreover, they can also be used to explain the global behavior of any model by examining the prototypes and the criticisms together with their predictions.

An example is shown in Figure 3.9 (Molnar, 2019). The top-left plot shows the data that consists of two features x_1 and x_2 , and the density estimation is illustrated with shaded background. Each of the other plots shows the different selections of prototypes, and the MMD2 values are shown in the titles. The prototypes are illustrated by large red dots and their distribution is shown using contour lines. The best selection of prototypes is shown in the bottom-left plot where the prototypes best cover the data and the MMD2 value is the lowest.

Advantages

The advantages of MMD include:

- MMD-critic can work with density estimates of a dataset.
- MMD-critic can be used to select both prototypes and criticisms, and finding criticisms is independent of the process of finding prototypes.

Disadvantages

The disadvantages of MMD include:

- Users need to specify the number of prototypes and criticisms, which introduces an additional parameter for the method.
- Users need to choose the kernel function and the kernel scaling parameter.
- All features are taken into consideration when computing MMD, disregarding the fact that some features might not be relevant for predicting.

3.5 Summary of explanation methods

In this section, we compare the above explanation methods using the following criteria:

- Explanation type. What is the explanation type? is it for global explanations or local explanations?
- Explanation representation. What is the form/representation of an explanation? Vector of weights, pattern, or instance?
- Model-agnostic. Whether the method is model-agnostic or not? Does the method need to know the internal structure/parameter of the models?
- Local linearity assumption. Does the method assume that the data is locally linearly separable?
- Support multi-explanation. Is the method able to generate multiple explanations for multi-explanation scenario?

The summary is shown in Table 3.1.

Method	Explanation type	Explanation representation	Model-agnostic?	Local linearity assumption?	Support multi-explanation?
Partial Dependence plot (PDP) (Friedman, 2001)	Global	Vector of weights	Yes	No	No
Individual Conditional Expectation (ICE) (Goldstein et al., 2015)	Local	Vector of weights	Yes	No	No
Feature Importance (Breiman, 2001) (Fisher, Rudin, and Dominici, 2018)	Local	Vector of weights	Yes	No	No
Shapley Values (Štrumbelj and Kononenko, 2014)	Local	Vector of weights	Yes	No	No
LIME (Ribeiro, Singh, and Guestrin, 2016c)	Local	Vector of weights	Yes	Yes	No
LORE (Guidotti et al., 2018), LACE (Pastor and Baralis, 2019)	Local	Pattern	Yes	No	No
Counterfactuals (Wachter, Mittelstadt, and Russell, 2017)	Local	Instance	Yes	No	Yes
Adversarial examples (Szegedy et al., 2013)	Local	Instance	No	No	Yes
Prototypes and criticisms (Kim, Khanna, and Koyejo, 2016)	Global	Instance	Yes	No	No

TABLE 3.1: Summary of explanation methods

Chapter 4

Pattern based Interpretable Feature Generation

4.1 Introduction

Interpretable feature engineering plays a key part in interpretable machine learning, and pattern-based feature engineering is an important aspect. This chapter introduces interpretable pattern-based feature engineering, and surveys different methods to generate interpretable features using patterns.

Feature engineering is the process of creating features for machine learning tasks. Feature transformation is an important method of feature engineering, and it involves transforming old features to new features in order to get more discriminative features. However, if the transformation is complex, the new features might not be interpreted easily. In this chapter, we investigate how frequent patterns can be used to generate discriminative and interpretable features.

Frequent patterns are combinations of feature values that have a high frequency of co-occurrence, which can represent interesting relationships among instances in the dataset. For example, one may find that milk and bread usually appear together in a store's transactions, and thus {milk, bread} is a frequent pattern. Frequent pattern mining is a classic problem in data mining research.

The intuition behind the idea of generating features using patterns is that feature combinations can be more discriminative than individual features (Cheng et al., 2007) (Dong et al., 1999) (Loekito and Bailey, 2009). For example, in the case of document classification, word phrases are more useful than single words. While exhaustive exploration of possible feature combinations is usually computationally infeasible, pattern based techniques provide a solution to reduce the search space for feature combinations and generate features that have good predictive power.

Besides the discriminative ability of patterns, they can also result in features that are highly interpretable. Compared with similar methods such as Principle Component Analysis (PCA) and other kernel methods, the interpretability of patterns enables users to reason about why a data mining system (e.g., a classifier) yields certain outcomes, especially when the outcomes contradict previous expectations.

Generating features using patterns generally follows three steps:

- Pattern mining. This addresses the question of how to generate patterns from a dataset.
- Pattern selection. This process selects representatives with high quality and removes useless patterns.

- Feature generation. This constructs a new feature space based on patterns. In the new pattern feature space, each instance is usually mapped to an m -dimensional vector, where m is the number of patterns.

This chapter is organized as follows: some definitions and preliminaries used in this chapter are represented in Section 4.2. The framework of pattern based feature generation is given in Section 4.3. Approaches to generating patterns are discussed in Section 4.4. Techniques to prune large pattern sets are given in Section 5.4.3. Strategies for constructing new features using patterns are discussed in Section 4.6. Applications of pattern based feature generation for classification and clustering are given in Section 4.7 and Section 4.8 respectively.

4.2 Preliminaries

In this section, the terminologies of pattern in the context other than vector/tuple data is given and discussed.

4.2.1 Patterns in transaction data

Frequent patterns are sometimes applied to transaction data, which assumes that there is a universal set of items in a given domain. Let $\mathcal{I} = \{I_1, I_2, \dots, I_m\}$ be the set of all items. A *transaction* T is a non-empty subset of $\mathcal{I}$ (each transaction may also be associated with a unique transaction identifier *tid*), and a dataset $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$ is a set of transactions.

A *pattern* p is also a non-empty subset of $\mathcal{I}$. The length of a pattern is defined by the number of items it contains. A pattern p is sometimes referred to as an l -pattern if its length is l . A transaction T matches (or satisfies) a pattern p if $p \subseteq T$.

The *matching dataset* of a pattern p in a dataset $\mathcal{T}$ is denoted as $mds(p, \mathcal{T})$, which is a subset of $\mathcal{T}$ where all transactions in the subset match p , and is formally defined as $mds(p, \mathcal{T}) = \{T \in \mathcal{T} | T \text{ matches } p\}$.

The *support* of a pattern in a dataset $\mathcal{T}$ is denoted as $support(p, \mathcal{T})$, which is equal to the proportion of transactions of $mds(p, \mathcal{T})$ in $\mathcal{T}$, and is formally defined as $support(p, \mathcal{T}) = \frac{|mds(p, \mathcal{T})|}{|\mathcal{T}|}$. A pattern p is called frequent if its support is greater than or equal to a user-specified threshold $minSupp$.

Table 4.1 depicts a synthetic transaction dataset with five transactions (T_1, T_2, T_3, T_4 and T_5). The first column shows the ID of each transaction, and the second column shows the items in the transaction. Table 4.2 depicts the frequent patterns mined from the above dataset with minimum support $minSupp = 0.6$. Each pattern has at least three transactions in its matching dataset. For example, pattern $p_4 = \{a, c\}$ is matched by T_1, T_2 and T_5 and thus its support is $3/5=0.6$.

TID	Items
T_1	a, b, c, d
T_2	a, c, e
T_3	b, c, d
T_4	a, e
T_5	a, b, c

TABLE 4.1: Synthetic transaction data

Pattern	Items	Matching dataset	support
p_1	a	T_1, T_2, T_4, T_5	0.8
p_2	b	T_1, T_3, T_5	0.6
p_3	c	T_1, T_2, T_3, T_5	0.8
p_4	a, c	T_1, T_2, T_5	0.6
p_5	b, c	T_1, T_3, T_5	0.6

TABLE 4.2: Frequent pattern set with threshold $\text{minSupp} = 0.6$

4.2.2 Connection with patterns in other format

Patterns for vector data

Vector/tuple data is commonly used in data mining tasks. For vector data, there is a pre-defined set $\{F_1, F_2, \dots, F_m\}$ of features/attributes, and each feature F_i is associated with a domain $\text{dom}(F_i)$, which could be numeric (continuous) or nominal (discrete). Table 4.3 gives an example of vector/tuple data. The dataset consists of three features F_1, F_2 and F_3 (TID helps to identify the instances and is not in the feature set), where F_1, F_2 are nominal and F_3 is numeric. In particular, $\text{dom}(F_1) = \{a, b, c\}$, $\text{dom}(F_2) = \{\text{yes}, \text{no}\}$ and $\text{dom}(F_3) = [1, 10]$.

As most pattern mining algorithms are designed for transactional data, vector data is usually transformed into transactional data with a discretization process for numeric features. For nominal features, the feature-value pair of an instance can be directly transformed to an item in the form of “feature=value”. For numeric features, the domain of these features is divided into a finite number of disjunctive intervals, and the feature-value pair of an instance is transformed into an item in the form “feature $\in$ interval”, where the value of the numeric feature must belong to the given interval. There are many discretization approaches, including the following commonly used approaches (Dong and Bailey, 2012): 1) equal-width, which divides the domain of a numeric feature into intervals of equal width, 2) equal-density, where all intervals have the same number of instances, and 3) entropy based discretization, which iteratively generates splitting points using information gain (Dougherty, Kohavi, and Sahami, 1995).

The transformation of the dataset shown in Table 4.3 to the corresponding transactional dataset is given in Table 4.4. Equal-width discretization is used for F_3 where the width is set to 5.

TID	F_1	F_2	F_3
T_1	a	yes	1
T_2	b	no	3
T_3	a	yes	5
T_4	c	yes	9

TABLE 4.3: Synthetic vector/tuple data

TID	Items
T_1	$F_1=a, F_2=\text{yes}, F_3 \in [1, 5)$
T_2	$F_1=b, F_2=\text{no}, F_3 \in [1, 5)$
T_3	$F_1=a, F_2=\text{yes}, F_3 \in [5, 10]$
T_4	$F_1=c, F_2=\text{yes}, F_3 \in [5, 10]$

TABLE 4.4: Corresponding transactional data

In the case of vector/tuple data types, a pattern is a conjunction of feature-value pairs (e.g., $\{F_1 = a \wedge F_3 \in [5, 10]\}$).

Patterns for sequence data

In sequence data, there is also a universal set of items denoted by $\mathcal{I} = \{I_1, I_2, \dots, I_m\}$. A *sequence* is an ordered list of itemsets, and it is denoted by $\langle s_1, s_2, \dots, s_l \rangle$, where s_i is an itemset. A pattern in sequence data is represented as a *subsequence*. A sequence $\langle a_1, a_2, \dots, a_l \rangle$ is a subsequence of $\langle b_1, b_2, \dots, b_n \rangle$ if there exist integers $1 \leq j_1 \leq j_2 \leq \dots \leq j_l \leq n$ such that $a_i \subseteq b_{j_i}$ for $i = 1, 2, \dots, l$.

Table 4.5 gives an example of a sequence dataset, where the set of items is $\{a, b, c, d, e, f, g\}$. Each sequence is associated with a sequence identifier SID. An itemset within a sequence is represented using brackets, and for brevity, when an itemset contains only one item, the brackets are omitted. The first sequence $\langle a(abc)(be) \rangle$ contains three itemsets: $a, (abc)$ and (be) . Sequence $\langle abe \rangle$ is a subsequence of S_1 , while $\langle ade \rangle$ is not.

SID	Items
S_1	$\langle a(abc)(be) \rangle$
S_2	$\langle a(ce)(bf)g \rangle$
S_3	$\langle (ab)(acd)(be) \rangle$
S_4	$\langle (be)(dg) \rangle$

TABLE 4.5: Synthetic sequence data

Patterns for graph data

A labeled graph G can be represented using a 4-tuple (V, E, α, β) , where V is a vertex set, $E \subseteq V \times V$ is an edge set, α is a function that assigns labels to vertices, and function β assigns label to edges. A pattern in the scenario of graph data is a *subgraph*. Given two labeled graphs $G = (V, E, \alpha, \beta)$ and $G' = (V', E', \alpha', \beta')$, G is a subgraph of G' iff

- $V \subseteq V'$,
- $\forall v \in V, \alpha(v) = \alpha'(v)$,
- $E \subseteq E'$,
- $\forall (u, v) \in E, \beta(u, v) = \beta'(u, v)$.

Contrast patterns

Frequent patterns are mined in an unsupervised manner, where no class information is considered. It is natural to extend this concept from a supervised perspective.

Suppose a dataset $\mathcal{T}$ is associated with classes/labels $C_1, C_2, \dots, C_k$, and $\mathcal{T}$ can be partitioned into k disjoint groups $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_k$ where all transactions in $\mathcal{T}_i$ have class C_i . A *contrast pattern* or *discriminative pattern* is a pattern that occurs frequently only in one subset with a certain class value while infrequently in the rest. In other words, the support of a contrast pattern differs significantly between the subset of interest and the rest of the dataset.

Assume dataset $\mathcal{T}_p$ labeled as C_p is the subset that is being contrasted or discriminated against the remaining subset $\mathcal{T}_n$, where $\mathcal{T}_n = \bigcup_{i \neq p} \mathcal{T}_i$. There are two common ways to define a contrast pattern: based on growth ratio or based on support delta.

The *growth ratio* of a pattern p is the ratio of the support of p in $\mathcal{T}_p$ to the support of p in $\mathcal{T}_n$: $GrRatio(p, \mathcal{T}_p, \mathcal{T}_n) = support(p, \mathcal{T}_p) / support(p, \mathcal{T}_n)$. The *support delta* of a pattern p is the difference of the support of p in $\mathcal{T}_p$ and the support of p in $\mathcal{T}_n$: $Supp_\delta(p, \mathcal{T}_p, \mathcal{T}_n) = support(p, \mathcal{T}_p) - support(p, \mathcal{T}_n)$. A pattern is called a contrast pattern or disriminative pattern if the chosen measure (growth ratio or support delta) is at least equal to a user-specified threshold. All discussions in this chapter will be using growth ratio to define a contrast pattern, and scenarios based on support delta extend in a similar way.

TID	F_1	F_2	F_3	Class
T_1	true	false	1	1
T_2	true	true	2	1
T_3	false	false	5	0
T_4	false	true	3	1

TABLE 4.6: Example dataset $\mathcal{T}$ for contrast patterns

Take the data set $\mathcal{T}$ shown in Table 4.6 as an example. Consider two patterns $p = \{F_1 = \text{true}, F_3 \leq 3\}$, $q = \{F_2 = \text{false}\}$. The matching data sets of p, q are $mds(p, \mathcal{T}) = \{T_1, T_2\}$, $mds(q, \mathcal{T}) = \{T_1, T_3\}$, and the corresponding supports are $supp(p, \mathcal{T}) = supp(q, \mathcal{T}) = \frac{2}{4} = 0.5$. Assume the data set is split into $\mathcal{T}_1 = \{T_1, T_2, T_4\}$ and $\mathcal{T}_2 = \{T_3\}$ by the class label, and the task is to find contrast patterns for $\mathcal{T}_1$. Then the growth ratios of p, q are $GrRatio(p, \mathcal{T}_1, \mathcal{T}_2) = \frac{2}{0} = +\infty$, and $GrRatio(q, \mathcal{T}_1, \mathcal{T}_2) = \frac{1}{1} = 1$. Pattern p is called a contrast pattern for $\mathcal{T}_1$ and q is not, if $minRatio$ is set to be greater than 1.

4.3 Framework of pattern based feature generation

The framework of pattern based generation follows three steps: 1) pattern mining, 2) pattern selection, and 3) feature generation. As shown in Figure 4.1, a pattern set Q is first mined from the given dataset. Then Q is pruned to a set P of representative patterns where insignificant and redundant patterns are removed. Finally, the remaining patterns are used to build a new feature space and the original dataset is mapped into the pattern feature space. The new feature space built on patterns is referred to as the *pattern space*.

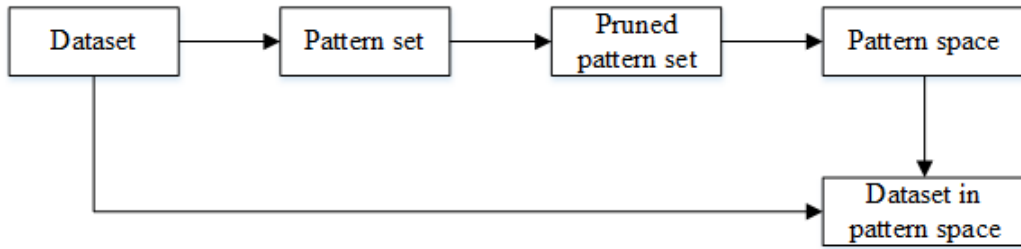


FIGURE 4.1: Framework of pattern based generation

An example of the framework is given in Figure 4.2. The original vector dataset contains n instances represented in feature space $\mathcal{F}^I = (F_1, F_2, \dots, F_d)$. A pattern set Q containing l patterns is first mined from the original dataset. Then the pattern set is pruned to a smaller set $P = \{p_1, p_2, \dots, p_m\}$, which contains the representative patterns with high quality according to user-specified measures. With the final pattern set P , the new pattern space $\mathcal{F}_p^m = (F_{p_1}, F_{p_2}, \dots, F_{p_m})$ is built based on these patterns. In the pattern space, each feature is a pattern and its feature value is calculated by a mapping function

$\phi_i(x)$, which maps an instance in the original feature space into the feature value for pattern p_i .

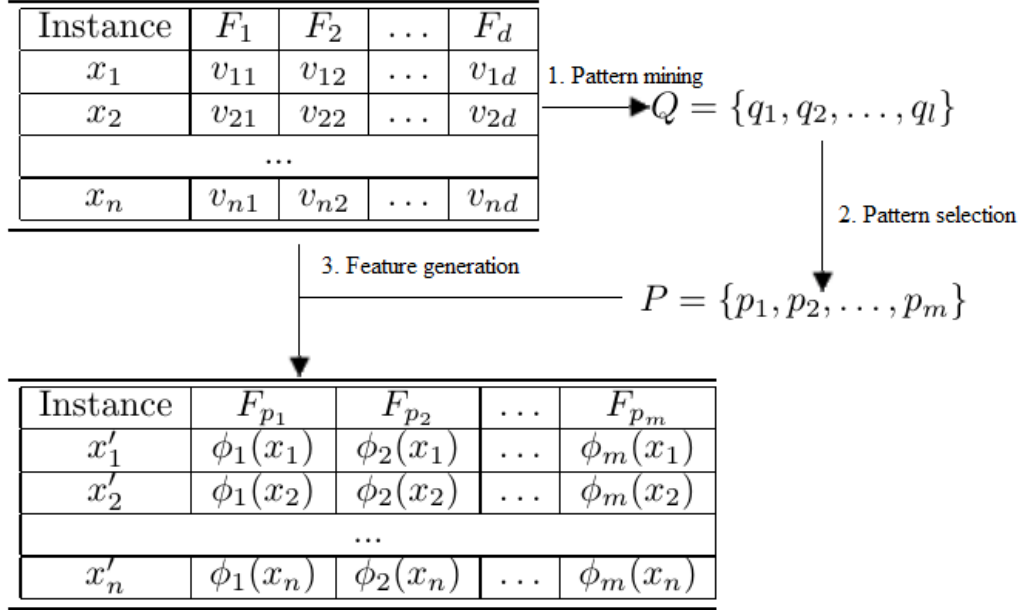


FIGURE 4.2: Framework of pattern based generation

For structural data types like sequence data and graph data, one follows the same framework to convert the structural data into vector data, such that existing vector-based algorithms (e.g., SVM) can be directly applied on the transformed data.

4.3.1 Pattern mining

Pattern mining is the initial step, where the inputs and outputs are described as follows:

- **Input:** $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$, $minSupp$, (optional) $minRatio$
- **Output:** $Q = \{q_1, q_2, \dots, q_l\}$

$\mathcal{T} = \{t_1, t_2, \dots, t_n\}$ is the given dataset, the common types include: 1) transactional data, 2) vector data, 3) sequence data, and 4) graph data. Transactional data and vector data are usually handled in the same way as they are mutually convertible. For sequence and graph data, extra information such as spatial distance should be taken into consideration.

$minSupp$ is the minimum support threshold. It is used to describe the degree to which a pattern is called frequent in a dataset. Although it is the only required parameter for most pattern mining algorithms, the quality of a pattern can be measured from other perspectives, such as information gain, given different domains. Thus a small $minSupp$ is usually desired to avoid information loss.

$minRatio$ is required in the case of contrast pattern mining. It is used to describe the degree to which a pattern is considered significant in a subset while insignificant in the rest of the dataset. Contrast patterns are more widely used for classification as they capture the relationship between features and class labels. The choice of $minRatio$ is similar to the choice of $minSupp$ and a small value is commonly used.

$Q = \{q_1, q_2, \dots, q_l\}$ is the output of mining process and is a set containing l patterns.

4.3.2 Pattern selection

As *minSupp* and *minRatio* are usually set to small values, the pattern set size is usually large and some of the patterns can be even noisy and useless. This motivates the problem of pruning the pattern set and selecting representative patterns for a given domain and task. The inputs and outputs are:

- **Input:** $Q = \{q_1, q_2, \dots, q_l\}, s(p), r(p_i, p_j), eval(PS), k$
- **Output:** $P = \{p_1, p_2, \dots, p_m\}$

$Q = \{q_1, q_2, \dots, q_l\}$ is a pattern set containing l patterns, which is the output from the previous mining process.

$s(p)$ defines a scoring function for individual patterns, which evaluates the significance of a pattern. Commonly used measures include support, growth ratio and information gain.

$r(p_i, p_j)$ evaluates the redundancy between two patterns p_i and p_j . It could be a distance function or similarity function such as l_1 norm distance or Jaccard similarity.

$eval(PS)$ measures the overall quality of a pattern set PS and it is usually the objective function for the final task such as classification or clustering. It helps to find an optimal or an approximately optimal pattern set in terms of the given tasks.

k is the constraint on the number of patterns, which means that at most k patterns are desired.

$s(p), r(p_i, p_j), eval(Q), k$ are all optional and the definitions of them depend on the specific domains or applications.

$P = \{p_1, p_2, \dots, p_m\}$ is the output of the reduced pattern set containing m patterns. If k is given, then $m \leq k$.

4.3.3 Feature generation

When the patterns are available, the next process creates new features based on these patterns, and maps the original data into the new feature space. There are two strategies to build the new feature space: 1) feature extraction, which creates new features using patterns, and 2) feature filtering, which selects a subset of the original features based on pattern information.

Feature generation using patterns creates new features and defines a mapping function from the original feature space to the new pattern space. Assume a dataset has d features and each instance in the dataset is a d -dimensional vector. Let $\mathcal{F}^d = (F_1, F_2, \dots, F_d)$ denote the original d -dimensional feature space. The pattern set $P = \{p_1, p_2, \dots, p_m\}$ is used to construct the new pattern space denoted by $\mathcal{F}_p^m = (F_{p_1}, F_{p_2}, \dots, F_{p_m})$. Then a mapping function $\Phi : x \mapsto x', x \in \mathcal{F}^d, x' \in \mathcal{F}_p^m$ is defined to transform the original d -dimensional instances into m -dimensional vectors in the pattern space, where each feature corresponds to a pattern. After the new pattern space is generated, it could either be used solely or combined with the original feature space for further analysis.

4.4 Pattern mining algorithms

Pattern mining is the first step for pattern based feature generation, and it provides a set of candidate patterns for the next process. This section presents common techniques to generate frequent patterns and contrast patterns. For each discussion, we first consider the cases for transactional data and vector data, then extend to the cases for sequence data and graph data where appropriate.

4.4.1 Frequent pattern mining

Common pattern mining algorithms can be categorized into three types and their variants: 1) Apriori, 2) FP-growth, and 3) Eclat. These three basic algorithms are discussed in this section.

Apriori

Apriori (Agrawal and Srikant, 1994) uses a level-wise search to find patterns, where l -patterns are used to find $(l + 1)$ -patterns. It starts with patterns with length 1, and all patterns in the initial set contain only one item. At each iteration, a pruning procedure is conducted to remove patterns that do not meet the minimum support threshold. The pruning process is based on the Apriori property: all non-empty subsets of a frequent pattern must also be frequent. It stops when no more patterns can be found. The detailed algorithm is given in Algorithm 1.

Algorithm 1: Apriori algorithm

Input: $\mathcal{T}, minSupp$

```

1  $P_1 = \{\text{length-1 frequent patterns}\};$ 
2  $c = 2;$ 
3 while  $P_{c-1} \neq \emptyset$  do
4    $Temp_c = \text{generate length-}c \text{ candidates from } P_{c-1};$ 
5   for every  $T$  in  $\mathcal{T}$  do
6     for every  $p$  in  $Temp_c$  do
7       if  $T$  matches  $p$  then
8          $p.count++;$ 
9       end
10    end
11  end
12   $P_c = \{p \in Temp_c | p.count \geq minSupp * |\mathcal{T}|\};$ 
13 end
14 Return  $\cup_c P_c;$ 
```

The inputs include the dataset $\mathcal{T}$ and the minimum support threshold $minSupp$. Line 2 generates all length-1 frequent patterns from the dataset as initial pattern set. Lines 4-10 repeatedly find length- c candidates from length- $(c - 1)$ frequent patterns and remove candidates whose supports are less than $minSupp$.

The main drawback of Apriori is that it needs to scan the entire dataset at every iteration. Many variants (Han et al., 2007) have been proposed to improve the efficiency from perspectives like hashing techniques (Park, Chen, and Yu, 1995), sampling techniques (Toivonen, 1996) and distributed mining (Agrawal and Shafer, 1996).

Sequence patterns also follows the Apriori property: any non-empty subsequence of a frequent sequence pattern must also be frequent. AprioriAll (Agrawal and Srikant, 1995) and GSP (Srikant and Agrawal, 1996) are two approaches that mine sequence patterns based on the Apriori property.

For graph data, typical Apriori-based techniques (Inokuchi, Washio, and Motoda, 2000)(Kuramochi and Karypis, 2001)(Vanetik, Gudes, and Shimony, 2002) discover graph patterns starting with small-size graphs and iteratively increase the size by 1 to generate larger graphs.

FP-growth

Frequent pattern growth (FP-growth) (Han, Pei, and Yin, 2000) first compresses a dataset into a frequent pattern tree (FP-tree) and then generates patterns directly from the tree. Unlike Apriori that repeatedly scans the whole dataset at each generation of l -patterns and generates a large number of candidates, FP-growth only scans the dataset twice and does not generate candidates that are not frequent.

Given the dataset $\mathcal{T}$ and the minimum support threshold $minSupp$, FP-growth is described in Algorithm 2.

Algorithm 2: FP-growth algorithm

Input: $\mathcal{T}, minSupp$

- 1 Scan the dataset and sort all length-1 patterns in frequency-descending order.;
 - 2 Construct a FP-tree. Start with a root node labeled as “null” and associated with a count 0. Then scan the dataset again. For every transaction, create a path containing all the items in the 1-patterns’ order. If the path $\mathcal{P}$ and some branch $\mathcal{B}$ in the tree share a common prefix, increase the counts of all nodes in the prefix by 1, and then create new branch using the remaining suffix of $\mathcal{P}$ and link it to the last node of $\mathcal{B}$ as a child.;
 - 3 $P = \{\}$. ;
 - 4 Generate patterns from the FP-tree. Start with the root node with prefix= $\{\}$, and recursively process the node as follows: concatenate the given prefix with the node being processed to form a new pattern and add it to P , then use the new pattern as the new prefix and pass it to its child nodes whose associated counts satisfy the threshold $minSupp$.;
 - 5 Return P .;
-

Experiments in (Han, Pei, and Yin, 2000) have shown that FP-growth is efficient and scalable, and is typically an order of magnitude faster than Apriori. Several variants have been proposed to improve FP-growth, and interested readers can refer to (Han et al., 2007).

FreeSpan(Han et al., 2000) and its extension PrefixSpan (Han et al., 2001) are the FP-Growth like approaches for sequence data. They mine sequence patterns by dividing the dataset into smaller groups and generating longer patterns from shorter patterns recursively.

For graph data, pattern-growth based algorithms maintain a frequent graph by adding new edges in all possible positions incrementally. gSpan(Yan and Han, 2002) avoids generating duplicates by performing depth-first search and using a right-most-extension technique.

Eclat

Both Apriori and FP-growth mine patterns from datasets in horizontal data format $\{tid : itemset\}$, where tid is the transaction identifier. Correspondingly, the datasets can be transformed into vertical data format $\{item : tid_set\}$. Elact (Zaki, 2000) mines patterns from vertical datasets.

First, the original dataset is transformed into a dataset in vertical form. Similar to Apriori, it repeatedly generates $(l + 1)$ -patterns from l -patterns using the Apriori principle, and terminates when no patterns can be found. Unlike Apriori, Eclat does not repeatedly scan the original datasets. However, if the tid_set can be quite long, then Eclat faces the same challenges as Apriori. It has also been proven that Eclat works for the

situation where a dataset contains many dense and long patterns (Han, Pei, and Kamber, 2011).

SPAD (Zaki, 2001) and SPAM (Ayres et al., 2002) transform horizontal sequence datasets into vertical format, and mine frequent sequence patterns based on Apriori property.

4.4.2 Contrast pattern mining

In this stage, the growth ratio is chosen as the threshold parameter to find contrast patterns. The use of other thresholds such as support delta follows a similar approach.

Indirect contrast pattern mining

An intuitive approach for contrast pattern mining is to mine frequent patterns first, then select patterns that meet the growth ratio threshold. The framework is described in Algorithm 3. The inputs include the frequent pattern set FP , the minimum growth ratio $minRatio$, the positive dataset $\mathcal{T}_p$ and the negative dataset $\mathcal{T}_n$. The procedure first calculates the growth ratios for every pattern and remove the patterns whose ratios are less than $minRatio$.

Algorithm 3: indirectContrastPatternMining

Input: $FP, minRatio, \mathcal{T}_p, \mathcal{T}_n$

```

1  $CP = \{\}$ ;
2 for every  $p$  in  $FP$  do
3    $p.ratio = \frac{support(p, \mathcal{T}_p)}{support(p, \mathcal{T}_n)}$ ;
4   if  $p.ratio \geq minRatio$  then
5      $CP = CP \cup \{p\}$ ;
6   end
7 end
8 Return  $CP$ ;
```

The methods proposed in (Cheng et al., 2007)(Deshpande et al., 2005)(Cong et al., 2005)(Dong et al., 1999)(Li, Han, and Pei, 2001)(Ma and Liu, 1998) all belong to this two-step framework. The major problem of two-step mining is its efficiency.

Direct contrast pattern mining

Direct contrast pattern mining directly discovers contrast patterns without generating a large number of candidates, thus it is more efficient than indirect contrast pattern mining.

The work in (Dong and Li, 1999) mines contrast patterns (which are called emerging patterns by the authors) by manipulating the borders (collection of closed patterns and generators).

The method proposed in (Bay and Pazzani, 2001) uses Apriori-like algorithms to generate contrast patterns. When pruning length- k candidates, it also prunes the patterns that does not meet the contrast criteria (support difference are used in this chapter). This method suffers the same problem as Apriori that needs to scan the whole dataset at every iteration.

DDPMine (Cheng et al., 2008) progressively reduces the data size by eliminating instances that are covered by found patterns. It generates one pattern at each iteration and removes the instances that match the pattern, and terminates when the remaining dataset is empty.

HARMONY (Wang and Karypis, 2005) uses an instance-centric approach to generate patterns. It ensures that every instance is associated with a pattern with the highest confidence.

DPClass (Shang et al., 2016) generates patterns using a random forest. It first constructs a random forest from the dataset, and then transforms the path from the root to a non-leaf node into a pattern for every tree in the forest. Although it is efficient even with large datasets, the search space is smaller comparing with other techniques.

In the scenario where transactions can be inserted and/or deleted over time, repeatedly running the mining algorithms is time-consuming. The works in (Li et al., 2004) (Bailey and Loekito, 2010) propose methods to incrementally generate contrast patterns.

For sequence data, a direct mining algorithm is proposed in (Ji, Bailey, and Dong, 2007), which generates patterns using depth-first search in a lexicographic sequence tree and checks the supports of patterns using an efficient bitset technique. This algorithm is efficient even with low support threshold.

For graph data, (Ting and Bailey, 2006) utilizes the connection of maximal common edge sets and the minimal contrast edge sets to generate contrast graph patterns.

Though direct contrast pattern mining avoids the generation of a complete frequent pattern set, it suffers the problem that some critical contrast patterns may be missing from the search space.

4.5 Pattern selection approaches

The choice of the minimum support threshold $minSupp$ has a major impact on the number of frequent patterns mined. The larger $minSupp$ is, the smaller the pattern set size is, and vice versa. In general, a relatively small $minSupp$ is chosen because a large $minSupp$ also results in considerable information loss during the mining process. Thus there are many frequent patterns, and exploiting all of them is redundant and usually computationally infeasible. This calls for pruning these patterns to reduce the number of patterns and choosing the best representatives to preserve the information in the dataset. There are two categories of techniques to prune a pattern set: post-processing pruning and in-processing pruning. Post-processing pruning techniques reduce the number of patterns after pattern candidates are generated. It is independent from the pattern mining algorithm, and thus can be employed for any existing mining algorithms. In-processing pruning techniques prune the pattern size during the mining process. It can avoid generating a large number of candidates, thus improving the efficiency.

4.5.1 Post-processing pruning

Post-processing pruning follows a pattern mining procedure that typically takes data and a minimum support threshold $minSupp$ as inputs, and outputs a set of pattern candidates denoted as P . The set P is then used as input to the post-processing pruning procedure, optionally together with a threshold k , which means that at most k patterns are desired. The goal of post-processing pruning is to remove weak patterns or redundant patterns and output a pruned pattern set Q , which is a subset of P . The definition of the strength or redundancy of a pattern varies in different domains or scenarios, and is discussed as follows.

Equivalence class

The concept of an *equivalence class* helps to select representatives and remove redundant patterns (Bayardo Jr, 1998) (Pasquier et al., 1999). The equivalence class of a pattern p in

a dataset $\mathcal{T}$ is the pattern set $EC(p) = \{q | mds(q, \mathcal{T}) = mds(p, \mathcal{T})\}$, where all patterns in $EC(p)$ have the same matching dataset as p . A pattern p is called a *closed pattern* if there exists no superset q of p such that $p \subset q$ and $q \in EC(p)$. Similarly, a pattern p is called a *generator* if there exists no subset q of p such that $q \subset p$ and $q \in EC(p)$. Closed patterns and generators are representatives of an equivalence class. An equivalence class contains one closed pattern and at least one generator.

TID	Items
T_1	a, b, c
T_2	a, b, c, d
T_3	a, b, c, e
T_4	b, g, f
T_5	a, d

TABLE 4.7: Dataset used to illustrate the equivalence class

Equivalence class	$\{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}$
Closed pattern	$\{a, b, c\}$
Generator	$\{c\}, \{a, b\}$

TABLE 4.8: Closed pattern and generator

Given the dataset shown in Table 4.7, an equivalence class is illustrated in Table 4.7. Patterns $\{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}$ are in the same equivalence class as the matching dataset of all of them is $\{T_1, T_2, T_3\}$. Pattern $\{a\}$ is not in this equivalence class as it is also matched by T_4 , which is not in the matching dataset of the others. Pattern $\{a, b, c\}$ is the closed pattern in this equivalence class and patterns $\{c\}, \{a, b\}$ are generators.

Though selecting closed patterns and/or generators reduces the pattern set size, the number of patterns mined for further analysis is still large especially when the support threshold is low. Pruning using equivalence class is usually used as the initial step, and closed patterns or generators are commonly selected as pattern candidates for further pruning.

Profile based techniques

A profile-based pruning method is proposed in (Yan et al., 2005). It first introduces a statistical model to deal with frequent patterns. Patterns are clustered into k clusters, then all patterns in each cluster are merged into one pattern profile. A probability distribution vector is used to represent a pattern profile. For item set $I = \{a_1, a_2, \dots, a_k\}$, a pattern profile is represented as $p = \{g(a_1), g(a_2), \dots, g(a_k)\}$, where $g(a_i)$ is the probability of item a_i occurring in the pattern. Moreover, it uses Kullback-Leibler (KL) divergence to measure the quality of a pattern. By using a probabilistic representation of patterns, it enables the use of a “fuzzy match” which relaxes the strict match that a pattern should be a subset of a transaction. When partitioning patterns into K clusters, it requires the patterns to be strongly correlated in both composition and support.

These pattern profiles are representatives of the pattern clusters, and depend heavily on the applied clustering algorithms (e.g., K means). The whole process requires good design of distance measures for clustering. The interpretability of pattern profiles as new patterns is another issue, as the profiles are just statistical counts of items.

Redundancy based techniques

A redundancy-aware method is proposed in (Xin et al., 2006). Quality measurement of pattern redundancy is based on the coverage of the pattern set on the data. This method maximizes the coverage of patterns (or minimizes the number of transactions that are not covered by any pattern). Similarly, another redundancy based method is KRIMP proposed in (Vreeken, Van Leeuwen, and Siebes, 2011), which applies the Minimal Description Length Principle (MDL) to find an optimal pattern set to cover the dataset best as well as to reduce the redundancy of patterns. The exact optimizations of these two methods are NP-hard, thus heuristic algorithms are used to approximate the optimal solution. Both methods avoid the problem that similar patterns are ranked closely and pruned results are not diversified. One drawback of these techniques is that they consider significance (e.g., support) and redundancy separately, thus the patterns mined can have low significance.

Probabilistic model based techniques

In (Wang and Parthasarathy, 2006), frequent patterns help to construct a graphical model in the form of a Markov Random Field (MRF). During the construction of a MRF, a pattern is pruned if it cannot introduce significant dependency to the current model.

The advantages of using a probabilistic model include: 1) fast inference using existing algorithms (e.g., Chow-Liu tree (Chow and Liu, 1968)), and 2) a very good representation. While pattern mining remains a challenge for high dimensional and time series data, probabilistic models can outperform traditional techniques. However, probabilistic models usually lack interpretability and assume the existence of prior distributions. Current research mainly focuses on using mined patterns to build the probabilistic model, where these patterns are still mined using classical algorithms.

4.5.2 In-processing pruning

Post-processing pruning can be treated as a two-step procedure, which filters patterns from a large number of candidates that are generated using traditional pattern mining algorithms. To avoid the generation of useless candidates, in-processing pruning techniques are proposed. This is a one-step pattern mining approach, where the pruning strategy is incorporated in the mining process.

Constraint based techniques

A constraint-based method is introduced in (Guns, Nijssen, and De Raedt, 2013). Two levels of constraints are used: local constraints (significance of a pattern, e.g., support, length) are imposed on individual patterns, together with global constraints (redundancy in a pattern set, e.g., distances among patterns, coverage on data set) on the whole pattern set. It has the advantage that it gives an optimal solution given two levels of constraints. However, it depends heavily on the constraint programming solver and does not scale well to large datasets. On the other hand, it still remains a challenge to design useful but relatively small constraints, as the quality and efficiency is mainly affected by the choice of constraints.

Bayesian theory based techniques

A classification-oriented technique KRSNB is proposed in (Gay and Boullé, 2012). Instead of having an unlimited growing pattern space, this method pre-defines a pattern

space, together with a prior distribution on this space. The process of pattern mining becomes extraction of local optima, and each local optima is a pattern. Then these patterns are used as classification rules.

The advantages of this method include: 1) It is a parameter-free method. 2) The resulting classifier has a competitive performance compared with other state-of-the-art pattern-based classifiers. 3) It is novel in the sense that it pre-defines a pattern space, which is different to traditional pattern mining methods. 4) It can apply directly to numeric data without requiring a discretization process.

However, it mainly suffers from problems that include: 1) The optimization process is heuristic and does not guarantee to find patterns that maximize the classification performance. 2) The performance depends heavily on the pre-defined pattern space.

Minimum Description Length (MDL) based techniques

KRIMP (Vreeken, Van Leeuwen, and Siebes, 2011) introduced earlier can also be categorized into this type of technique. SLIM (Smets and Vreeken, 2012) is an MDL based method that directly applies to the data, instead of an existing pattern set. This is also the main difference between KRIMP and SLIM. A code table is proposed to summarize the pattern space, then heuristic optimization using branch-and-bound is applied to mine good patterns following the principle of selecting highly frequent patterns with a short length.

SLIM's main advantages are: 1) It does not need to mine pattern sets using other techniques and can be directly applied to the data, which is efficient compared with post-processing methods. 2) It is parameter free. 3) It guarantees a good coverage of the data. Meanwhile, SLIM is computationally expensive, and only performs well on small datasets. When using SLIM on large datasets, given a limited time, early termination is very likely to generate patterns of poor quality.

4.6 Pattern based feature generation

When the patterns are ready, the next step is then to create new features based on these patterns, and map the original data into the new feature space. Given a dataset $X = \{x_1, x_2, \dots, x_n\}$, where x_i is a d -dimensional vector and a pattern set $P = \{p_1, p_2, \dots, p_m\}$, a *mapping function* $\Phi(x)$ is defined to transform the original d -dimensional instances into m -dimensional vectors in the pattern feature space, where each feature corresponds to a pattern. Let $\mathcal{F}^d = (F_1, F_2, \dots, F_d)$ denote the original d -dimensional feature space, and $\mathcal{F}_p^m = (F_{p_1}, F_{p_2}, \dots, F_{p_m})$ denote the new pattern space. Then a mapping function $\Phi : x \mapsto x', x \in \mathcal{F}^d, x' \in \mathcal{F}_p^m$ is defined as $\Phi(x) = (\phi_1(x), \phi_2(x), \dots, \phi_m(x))$, where $\phi_m(x)$ calculates the feature value.

4.6.1 Unsupervised mapping functions

Binary mapping function

Figure 4.3 gives an example of generating new features using patterns. The original data contains three features *gender*, *age* and *pclss*, and the pattern set contains two patterns p_1, p_2 . Thus the new feature space consists of two features corresponding to p_1, p_2 . For an instance x , the i th value of the corresponding vector in the pattern space is 1 if x matches p_i , and 0 otherwise. The first instance matches p_1 but does not match p_2 , thus the new instance is $(p_1 = 1, p_2 = 0)$. Similarly, instance 2 matches p_2 but does not match p_1 , thus the new instance is $(p_1 = 0, p_2 = 1)$.

instance	gender	age	pclass
1	male	20	1st
2	female	12	3rd

$$\begin{array}{c}
+ \\
p_1 = (\text{gender} = \text{male}, \text{age} \geq 18) \\
p_2 = (\text{pclass} = 3\text{rd})
\end{array}
=
\begin{array}{c}
\begin{array}{c|c|c}
\text{instance} & p_1 & p_2 \\
\hline
1 & 1 & 0 \\
2 & 0 & 1
\end{array}
\end{array}$$

FIGURE 4.3: Constructing a pattern space using binary mapping function

In the above example, the mapping function is a simple binary function. Formally, given a dataset $X = \{x_1, x_2, \dots, x_n\}$ and a pattern set $P = \{p_1, p_2, \dots, p_m\}$, the mapping function is defined as $\Phi(x) = (\phi_1(x), \phi_2(x), \dots, \phi_m(x))$, where

$$\phi_i(x) = \begin{cases} 1, & \text{if } x \text{ matches } p_i \\ 0, & \text{otherwise} \end{cases}$$

Binary mapping functions are the most widely used mapping function for pattern based feature generation. PatClass (Cheng et al., 2007) and DDPMine (Cheng et al., 2008) use binary mapping functions to calculate the new values in the pattern space for vector data. The work in (Zaïane et al., 2006) uses the binary mapping function for sequence classification. The study in (Fei and Huan, 2008) transforms a set of graphs to vectors using a binary mapping function.

Frequency based mapping function

Binary mapping functions treat all patterns as equally important features, which does not hold for many cases. Patterns are either in good quality or poor quality given a user-specified scoring function, and it is natural to use these measures as feature values in the pattern space.

Support or frequency can be used as the indicator of pattern quality. The mapping function based on frequency is defined as $\Phi(x) = (\phi_1(x), \phi_2(x), \dots, \phi_m(x))$, where

$$\phi_i(x) = \begin{cases} \text{support}(p_i), & \text{if } x \text{ matches } p_i \\ 0, & \text{otherwise} \end{cases}$$

4.6.2 Supervised mapping functions

Confidence based mapping function

When using contrast patterns to build the new feature space, one can evaluate the confidence of a pattern and use it as a new feature value. Given a dataset $\mathcal{T}$ and a contrast pattern p that frequently occurs in a subset with class label c , the confidence of the pattern is

$$\text{confidence}(p) = \frac{\text{support}(p, \mathcal{T}_c)}{\text{support}(p, \mathcal{T})}$$

where $\mathcal{T}_c$ is the subset that contains all instances with class c in $\mathcal{T}$.

A mapping function using confidence is defined as $\Phi(x) = (\phi_1(x), \phi_2(x), \dots, \phi_m(x))$, where

$$\phi_i(x) = \begin{cases} \text{confidence}(p_i), & \text{if } x \text{ matches } p_i \\ 0, & \text{otherwise} \end{cases}$$

Growth ratio based mapping function

Similar to confidence, growth ratio is another indicator of the predictive power of a pattern. A mapping function using growth ratio is defined as $\Phi(x) = (\phi_1(x), \phi_2(x), \dots, \phi_m(x))$, where

$$\phi_i(x) = \begin{cases} \text{GrRatio}(p_i), & \text{if } x \text{ matches } p_i \\ 0, & \text{otherwise} \end{cases}$$

The definition of growth ratio $\text{GrRatio}(p_i)$ can be found in Section 4.2.2. Similarly, one can also use support delta as alternative to growth ratio, e.g., the work in (Huan et al., 2004) uses support delta to measure the discriminative power of a pattern feature.

Chi-square based mapping function

The contingency table of a contrast pattern p associated with class label c is shown in Table 4.9. n_{11} is the number of instances that match p and are labelled as c ; n_{12} is the number of instances that match p but are not labelled as c ; n_{21} is the number of instances that do not match p but are labelled as c ; n_{22} is the number of instances that do not match p and also are not labelled as c .

	c	$\neg c$	Total
p	n_{11}	n_{12}	a_1
$\neg p$	n_{21}	n_{22}	a_2
Total	$ \mathcal{T}_c = b_1$	$ \mathcal{T}_{\neg c} = b_2$	$ \mathcal{T} = N$

TABLE 4.9: The contingency table for pattern p in dataset $\mathcal{T}$

Given the contingency table of a pattern p , now we can define chi-square χ^2 for the pattern:

$$\chi^2(p) = \sum_{i=1}^2 \sum_{j=1}^2 \frac{(n_{ij} - E_{ij})^2}{E_{ij}}$$

where $E_{ij} = \frac{(\sum_{k=1}^2 n_{ik}) \times (\sum_{k=1}^2 n_{kj})}{N}$ is the expected frequency count in the corresponding cell of the contingency table.

Then the mapping function using χ^2 is defined as $\Phi(x) = (\phi_1(x), \phi_2(x), \dots, \phi_m(x))$, where

$$\phi_i(x) = \begin{cases} \chi^2(p_i), & \text{if } x \text{ matches } p_i \\ 0, & \text{otherwise} \end{cases}$$

It is pointed out in CMAR (Li, Han, and Pei, 2001) that a higher χ^2 value may favor minority classes, and the authors propose to use a weighted χ^2 measure to address the issue. Interested readers can refer to (Li, Han, and Pei, 2001).

Hybrid mapping function

The mapping function can also be defined from combinations of different criteria. For example, CAEP (Dong et al., 1999) and work in (Ramamohanarao, Bailey, and Fan,

2005) measure the pattern quality from both frequency and growth ratio. Similarly, the mapping function can be defined as $\Phi(x) = (\phi_1(x), \phi_2(x), \dots, \phi_m(x))$, where

$$\phi_i(x) = \begin{cases} \frac{GrRatio(p_i)}{GrRatio(p_i)+1} * support(p_i), & \text{if } x \text{ matches } p_i \\ 0, & \text{otherwise} \end{cases}$$

4.6.3 Feature generation for sequence data and graph data

The mapping functions discussed above can be directly applied on sequence data and graph data. The process transforms the structured sequences/graphs into vector format, where each sequence/graph is converted to a vector of feature values. Such transformation helps bridge the gap between vector-data-based techniques and sequence/graph data, e.g., an SVM can be directly applied on a transformed graph dataset.

There may be some special considerations for sequence data and graph data, e.g., the spatial distribution of a subsequence/subgraph can carry important information. Here we discuss a solution to build the mapping function using structural information from sequence/graphs.

For example, in sequence data, an important property can be how spread out the subsequences are. This can be defined as a function to measure how many gaps are interspersed within a subsequence. For example, given two strings “fare” and “father”, “far” occurs as subsequence of both, but we may consider that its occurrence in “fare” is more important as it is contiguous.

Given a sequence s and a subsequence pattern p , let $l(s, p)$ denote the length of the minimal contiguous sequence that contains p in s . For example, given the subsequence “far”, “far” is the corresponding minimal contiguous sequence in “fare” thus the length is 3, and “father” is the corresponding minimal contiguous sequence in “father” thus the length is 6.

Now, we can define the mapping function as $\Phi(x) = (\phi_1(x), \phi_2(x), \dots, \phi_m(x))$, where

$$\phi_i(x) = \begin{cases} l(x, p_i), & \text{if } x \text{ matches } p_i \\ 0, & \text{otherwise} \end{cases}$$

4.6.4 Comparison with similar techniques

One may consider the difference between frequent patterns and similar methods such as Principle Component Analysis (PCA) and Linear Discriminant Analysis (LDA) that select or construct new features from the original features. PCA produces a new feature space where features are ranked according to the amount of variance they capture in the original feature space, but there is no guarantee that the components also have good discriminative power and new generated features may not be interpretable. LDA aims to reduce the dimensionality of the original features while preserving as much information as possible, but it fails to capture relationships (e.g., correlations) among instances that belong to the same class.

Kernel methods also create new feature spaces and map data into those new spaces. However, designing good kernel functions is not easy and it usually depends on domain knowledge. The interpretability of newly generated features is another issue, as in most cases they cannot even be understood by domain experts.

Spectrum kernel methods for sequence data and graph data are closely connected to pattern based feature generation. They differ in the sense that spectrum kernel methods enumerate all possible subsequences/subgraphs up to a maximum length, and do not take

the discriminative power of each subsequence/subgraph into consideration. Also, generating all possible subsequences/subtrees is computationally expensive and can require significant tuning afterwards.

4.7 Pattern based feature generation for classification

Classification is a major topic in data mining. In a typical classification problem, a classifier is built on a training dataset, where the class information is known, then the classifier is able to give predictions for unseen data where the class information is unknown.

This section presents existing solutions that employ patterns for classification. The approaches fall into two categories: 1) direct classification in the pattern space, which makes predictions directly from vector value of instances in the pattern space, and 2) indirect classification in the pattern space, which trains a classifier on the transformed dataset and makes prediction using that classifier.

4.7.1 Problem statement

Given a dataset $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$ associated with label set C in feature space $\mathcal{F} = (F_1, F_2, \dots, F_d)$ shown in Table 4.10, and a pattern set $P = \{p_1, p_2, \dots, p_m\}$ with new feature space $\mathcal{F}_p^m = (F_{p_1}, F_{p_2}, \dots, F_{p_m})$ shown in Table 4.11, $\mathcal{T}$ is transformed into $\mathcal{T}' = \{t'_1, t'_2, \dots, t'_n\}$ using a mapping function $\Phi(t) = (\phi_1(t), \phi_2(t), \dots, \phi_m(t))$. **Classification in the pattern space** predicts the class label of an unseen instance t_{new} using its projection in the pattern space.

	F_1	F_2	$\dots$	F_d	C
t_1	v_{11}	v_{12}	$\dots$	v_{1n}	c_1
t_2	v_{21}	v_{22}	$\dots$	v_{2n}	c_2
t_3	v_{31}	v_{32}	$\dots$	v_{3n}	c_3
$\dots$					

TABLE 4.10: Dataset in original feature space

	F_{p_1}	F_{p_2}	$\dots$	F_{p_m}	C
t'_1	$\phi_1(t_1)$	$\phi_2(t_1)$	$\dots$	$\phi_m(t_1)$	c_1
t'_2	$\phi_1(t_2)$	$\phi_2(t_2)$	$\dots$	$\phi_m(t_2)$	c_2
t'_3	$\phi_1(t_3)$	$\phi_2(t_3)$	$\dots$	$\phi_m(t_3)$	c_3
$\dots$					

TABLE 4.11: Dataset in pattern space

4.7.2 Direct classification in the pattern space

Every feature in the pattern space can be directly assigned with a class label, e.g., the class that is associated with a contrast pattern. Let c_i denote the class label associated with pattern p_i . Given an unseen instance t_{new} and $t'_{new} = (\phi_1(t_{new}), \phi_2(t_{new}), \dots, \phi_m(t_{new}))$ in the pattern space, the prediction for t_{new} can be made directly from the feature values of t'_{new} .

The pattern-class $\langle p_i, c_i \rangle$ pair in the pattern space means that if pattern p_i is satisfied then the class label is c_i . Such rules provide **IF...THEN...** reasoning for classification, and can be easily understood by users. Many recent works suggest that such rule based

classifiers have the potential to be highly interpretable (Caruana et al., 2015) (Kohavi, 1995) (Letham et al., 2015) (Wang and Rudin, 2015), which is important for the user to decide whether a model or prediction is trustworthy or not

Prediction by most probable pattern

One solution is to rank the feature values in t'_{new} and find the feature F_{p_i} with the highest priority, and then assign the class label c_i associated with p_i to t_{new} . Formally:

$$ind = \underset{i}{\operatorname{argmax}}(\phi_i(t_{new}))$$

$$prediction(t_{new}) = c_{ind}$$

In the case that the values are all 0s in t'_{new} , a default class label is generally desired, and the default class is usually set as the class label that is associated with most instances in the training dataset.

If there are multiple patterns/features have the same highest value, we can randomly choose one as proposed in CBA (Ma and Liu, 1998).

CBA, CAEP (Dong et al., 1999) and iCAEP (Zhang, Dong, and Ramamohanarao, 2001) are implementations of using only one pattern for direct classification.

Prediction by pattern set

We can also use the feature values as weights of the corresponding patterns, and compute the weighted votes for each class. This technique leverages the contributions of all patterns.

To predict an unseen instance t_{new} , it first creates a k -dimensional vector $vote$, where k is number of class labels, and then calculates the votes for each class using feature values in the pattern space:

$$vote[i] = \sum_{j:c_j=i} (\phi_j(t_{new}))$$

for $i = 1, 2, \dots, k$. Then the class label with the highest vote is returned as the prediction for t_{new} .

CMAR (Li, Han, and Pei, 2001), CPAR (Yin and Han, 2003), JEP-C (Li, Dong, and Ramamohanarao, 2001) and HARMONY (Wang and Karypis, 2005) all follow this framework for vector data classification. The method in (Zhou, Cule, and Goethals, 2016) is an implementation for sequence data.

4.7.3 Indirect classification in the pattern space

Though making prediction directly from values in the pattern space is easily interpretable, the accuracy is still a bottleneck for this category of techniques. Another approach achieves better performance at the cost of less interpretability. It is indirect classification that builds a new classifier (it could be any classifier model) on the transformed data and makes predictions using the trained model.

The framework of the training process is shown in Algorithm 4. Given the training dataset $\mathcal{T}$ and a mapping function $\Phi(t)$, all the training instances are transformed into new instances in the pattern space using the mapping function, and a user-specified classifier is trained on the new instances.

The framework of predicting an unseen instance is described in Algorithm 5. The unseen instance is first projected into the pattern space using the same mapping function used in the training process, and then predicted using the trained classifier cl .

Algorithm 4: indirectClassification

Input: $cl, \mathcal{T}, \Phi(t)$

```

1  $T' = \{\}$ ;
2 for every  $t$  in  $\mathcal{T}$  do
3    $t' = \Phi(t)$ ;
4    $T' = T' \cup \{t'\}$ ;
5 end
6  $cl.train(T')$ ;
7 Return  $cl$ ;
```

Algorithm 5: indirectClassification_Predict

Input: $cl, t, \Phi(t)$

```

1  $t' = \Phi(t)$ ;
2 Return  $cl.predict(t')$ ;
```

PatClass(Cheng et al., 2007) and DDPMine (Cheng et al., 2008) belong to this framework for vector data. The work in (Zaiane et al., 2006) gives an implementation for sequence data, and (Huan, Wang, and Prins, 2003) (Fei and Huan, 2008) are proposed for graph data.

For vector data, we can combine the original feature space and pattern space into a larger feature space if the dimensionality is not a concern. The combined feature space preserves the information of the original dataset and adds extra features that are discriminative.

4.7.4 Connection with stacking technique

Stacking method (Wolpert, 1992) trains a combiner classifier on predictions of several other classifiers. Several classifiers (that could be different types, or different parameter settings of the same type) are trained on the given dataset first, then a combiner classifier is trained to make predictions using all predictions of other classifiers as inputs.

Classification in the pattern space belongs to the framework of stacking techniques, as all the pattern features can be treated as special classifiers that always output single labels.

4.8 Pattern based feature generation for clustering

The goal of clustering is to divide the data into groups where instances in the same group are similar to each other. Most clustering techniques require the notion of pairwise similarity, e.g., a distance function, which is crucial for their performance. The design of a good distance function usually depends on prior domain knowledge. Recent works (Aggarwal and Han, 2014) (Dong and Taslimitehrani, 2015) have shown that frequent patterns can be used to help define the similarity between two instances, especially for high-dimensional data where distance is not very meaningful (Beyer et al., 1999).

4.8.1 Clustering in the pattern space

One of the key steps in clustering is to measure pair-wise similarity between instances in a dataset. While the definition of similarity differs in different data types, transforming the original dataset into the pattern space provides uniform measures in terms of vector

values in the pattern space, e.g., (Guralnik and Karypis, 2001) measures similarity of two sequences in the pattern space using cosine distance:

$$\text{sim}(s, t) = \frac{s' \cdot t'}{\|s'\| \|t'\|}$$

where s, t are two sequences, and s', t' are vectors that are projections of s, t in the pattern space.

Another commonly used measure is Hamming distance for vectors in the pattern space:

$$\text{sim}(s, t) = \sum_{i=1}^m |s_i - t_i|$$

where s, t are two sequences, and s', t' are vectors that are projections of s, t in the pattern space.

Clustering in the pattern space is similar to indirect classification in the pattern space, where the given original dataset (which could be transactions, vectors, sequences, graphs) is transformed into vectors in the pattern space, then traditional clustering algorithms (e.g., K-means) are applied on the new dataset (Guralnik and Karypis, 2001) (Wang et al., 2007).

4.8.2 Subspace clustering

In subspace clustering, no pattern feature space is generated, and patterns are used to filter the original features. The basic idea of subspace clustering is that two instances may not be similar over all attributes, but they are similar over a subset of attributes. A subset of attributes is referred as a subspace. Defining similarity on the subspace can avoid the influences of “irrelevant attributes”. Exhaustive exploration of all possible subspaces is computationally infeasible as there exist $2^d - 1$ subspaces for a d -dimensional dataset. Frequent pattern mining is one approach to help generate subspaces. Reference (Aggarwal and Han, 2014) gives the connection between frequent pattern mining and subspace search as illustrated in Table 4.12.

Frequent pattern mining	Subspace search
Item	Dimension (attribute)
Pattern	Subspace (set of attributes)
Frequent patterns	‘Interesting’ subspace

TABLE 4.12: Connection between frequent pattern mining and subspace search

Suppose a pattern p is mined as a subspace. Traditional similarity measures such as Euclidean distance and Jaccard distance can be applied on the attributes that occur in pattern p . For example, given two instances $x_1 = \{a, b, c, f\}$, $x_2 = \{b, c, d, e\}$ and a pattern $p = \{b, d\}$, and two instances are mapped to the pattern subspace as $x'_1 = \{b\}$, $x'_2 = \{b, d\}$, then the Jaccard distance of x_1, x_2 on subspace p is $\text{Jaccard}_p(x_1, x_2) = \text{Jaccard}(x'_1, x'_2) = \frac{|x'_1 \cap x'_2|}{|x'_1 \cup x'_2|} = \frac{1}{2}$.

4.9 Conclusion

This chapter has discussed the use of frequent patterns as features for data mining tasks. Frequent patterns are interpretable and have the potential to be highly discriminative features. Techniques to generate patterns and to prune a large pattern set were first discussed. Then we described techniques that employ frequent patterns to generate features for classification and clustering. We have also discussed scenarios where the underlying data type is a sequence or graph.

There are still many open questions in this area:

1. Most mapping functions evaluate the feature values in the pattern space solely by the significance of patterns, which can result in the domains of pattern features containing only two values: the significance of patterns and 0. This motivates the research direction to develop instance-level mapping functions that leverage the significance of patterns and that of instances. The gap based mapping function for sequence data is an example where the same pattern can exhibit different levels of importance for different instances.
2. Pattern mining techniques have been developed and applied in complex data types such as spatio-temporal data and image data. It is natural to extend the framework of pattern based feature generation to new data types as well, especially in an efficient way.
3. Interpretability has drawn more and more attention recently. Pattern based techniques are a promising class of data mining methods which are easy to interpret, while having the capacity to be highly discriminative. It remains a challenge to quantify the interpretability of patterns, and investigate the relationship between interpretability and performance (e.g., accuracy for classification) of patterns.
4. The framework of pattern based feature generation can be divided into three individual steps: pattern mining, pattern selection and feature generation. In many cases, these three steps are executed separately, which means that the objectives of pattern mining and pruning may not be related to feature generation. It is interesting to see how a pattern mining or pruning approach could directly consider feature generation as the objective.

Chapter 5

Exploiting patterns to explain individual predictions

5.1 Introduction

This chapter proposes an explanation method called Pattern Aided Local Explanation (PALEX) to provide instance-level explanations for any classifier. PALEX takes a classifier, a test instance and a frequent pattern set summarizing the training data of the classifier as inputs, then outputs the supporting evidence that the classifier considers important for the prediction of the instance. To study the local behavior of a classifier in the vicinity of the test instance, PALEX uses the frequent pattern set from the training data as an extra input to guide generation of new synthetic samples in the vicinity of the test instance. Contrast patterns are also used in PALEX to identify locally discriminative features in the vicinity of a test instance. PALEX is particularly effective for scenarios where there exist multiple explanations. In our experiments, we compare PALEX to several state of the art explanation methods over a range of benchmark datasets and find that it can identify explanations with both high precision and high recall.

It is desirable that classifiers are both interpretable and accurate. However, in most cases users have to balance accuracy and interpretability when they select a model. The challenge comes from the fact that complex models (e.g., random forests and neural networks) tend to be more accurate but less interpretable, while simple models (e.g., decision trees and logistic regression) tend to be less accurate but more interpretable. This raises the problem: can we increase the interpretability of complex models without sacrificing their accuracy? In situations where interpretability is about the ability to provide explanations, we can use model-agnostic explanation methods.

Early works (Craven and Shavlik, 1996) (Kurfess, 2000) (Barakat and Diederich, 2005) (Fung, Sandilya, and Rao, 2005) (Martens et al., 2007) on providing explanations recognize the interpretability of patterns that are conjunctions of feature-value conditions, and apply pattern based methods to explain complex models, such as neural networks and SVMs, at the global model level. This category of explanations is called **global explanations**, as it involves descriptions of the whole model. However, global fidelity does not necessarily imply local fidelity (Ribeiro, Singh, and Guestrin, 2016c), and global explanations may not be very useful for understanding an individual prediction.

Patterns have been widely adopted in the interpretability research community as they are intrinsically easy to interpret. Recent works (Robnik-Šikonja and Kononenko, 2008) (Adler et al., 2016) (Wang et al., 2016) (Ribeiro, Singh, and Guestrin, 2016c) on instance-level **local explanations** for individual predictions investigate the feature contributions and generate a vector of weights as the form of an explanation. However, there are drawbacks to this type of method: (1) A representation using a vector of weights for an explanations is less precise, as the meaning of the weights can be ambiguous depending on

the feature scales, which means that the same weight value could have different interpretations for different features that use different value ranges (i.e., when a feature is in the range of $[0, 100]$ and another feature in the range of $[0, 50000]$). (2) These methods only produce a single explanation for a prediction, and are not suitable for situations where there exist more than one reasonable explanation for a single prediction (e.g., in a disjunctive normal form where the result is true, there might be more than one true clause).

When extracting local explanations, existing methods typically generate a set of neighbour instances of a test instance, and then find local explanations from the generated neighbours given a classifier. The quality of the explanations depends heavily on the quality of the generated neighbourhood. In order to precisely find the local explanation for a test instance, the generated neighbour instances should be representative and contain sufficient information about the local behavior of the classifier (i.e., the local classification boundary). Unlike most techniques that use standard Euclidean distance to assess the proximity of a test instance to its neighbours, we explore the use of patterns to assess the proximity. Another key aspect is to extract explanations that reveal why the classifier makes a given prediction about the test instance. For this task, we employ contrast patterns to identify discriminative features for the given test instance.

This chapter proposes a pattern aided method (PALEX) to find **supporting evidence** for the predictions of any classifier. The main contributions include:

- PALEX exploits frequent patterns for discovering local explanations for a test prediction. In particular, it uses frequent patterns to better guide the generation of a local neighbourhood around a test instance, rather than randomly generating it. This neighbourhood is then used as the basis for formulating explanations.
- PALEX extracts contrast patterns as explanations, finding the discriminative features for the class of an individual instance based on the local neighbourhood. It does not make any assumption about local linearity.
- PALEX enriches explanation power by providing multiple explanations for an individual prediction.

5.2 Related Work

Pattern based interpretable models

One category of classifiers, called interpretable models, are inherently able to provide explanations for their predictions. Since patterns represented as sets of attribute-value pairs can be easily understood by users with or without a data mining background, pattern based classifiers (Caruana et al., 2015) (Kohavi, 1995) (Letham et al., 2015) (Wang and Rudin, 2015) are a promising class of data mining techniques that are easy to interpret. For example, the path from the root to the leaf of a decision tree for a given instance can be used to explain the prediction, or a matching rule in the rule list can also be used as the explanation. The goal of this research field is to develop models that are interpretable. For such models, there is always a trade-off between interpretability and accuracy (Freitas, 2014), and pattern based interpretable classifiers usually have lower accuracy than state-of-the-art classifiers like random forests and neural networks. Thus it still remains a challenge to achieve both high interpretability and high accuracy. Our proposed method PALEX recognizes the interpretability of patterns and suggests a model-agnostic way to help provide explanations such that users can choose any classifier without sacrificing the interpretability.

Global explanations

A common model-agnostic approach is to extract post-hoc explanations at the model-level (Ribeiro, Singh, and Guestrin, 2016a). Early works trained pattern based models or extracted patterns to mimic black-box classifiers. Craven et al. (Craven and Shavlik, 1996) build decision trees and Kurfess (Kurfess, 2000) extract patterns to approximate neural networks. They re-label the training data using a trained network and then learn a decision tree or extract patterns on new training data. Similarly, Barakat et al. (Barakat and Diederich, 2005), Fung et al. (Fung, Sandilya, and Rao, 2005) and Martens et al. (Martens et al., 2007) extract patterns for SVMs from the re-labeled training data. Instead of interpreting the behaviour of the whole classifier, SCaPE (Duivesteijn and Thaele, 2014) uses patterns to summarize the characteristics of the subgroup of data on which the classifier is likely to make incorrect predictions. GoldenEye++ (Henelius et al., 2015) explains a classifier from the view of interacting attributes by grouping the attributes exploited by the classifier.

Explanations provided by these methods are model-level global explanations. As pointed out in (Ribeiro, Singh, and Guestrin, 2016c), global fidelity does not imply local fidelity, as features that are important at the model level are not necessarily important at an instance level. Our method proposed in this chapter, utilizes this interpretability of patterns which are easily understandable by humans, and identifies local explanations represented using contrast patterns at the instance level.

Local explanations

Recent works have focused on instance-level local explanations, which are explanations for individual predictions. Robnik et al. (Robnik-Šikonja and Kononenko, 2008), Henelius et al. (Henelius et al., 2014), Adler et al. (Adler et al., 2016), and (Koh and Liang, 2017) compute a vector of weights representing the predictive powers of attributes. Parzen (Baehrens et al., 2010) calculates the gradient of the predictor with respect to a given instance to explain its prediction, and Selvaraju et al. (Selvaraju et al., 2017), and Wang et al. (Wang et al., 2016) similarly compute the gradient as an image mask.

LIME (Ribeiro, Singh, and Guestrin, 2016c) trains a local linear interpretable model in the vicinity of an instance and uses this new model to explain the instance. aLIME (Ribeiro, Singh, and Guestrin, 2016b) (Ribeiro, Singh, and Guestrin, 2018) is a variation of LIME, and is the first method to use patterns (which are referred to as “anchor” rules in aLIME) as local explanations. Both methods use extra data preprocessing steps (i.e., normalization) to calculate the distance function when assessing the vicinity of an instance. Our proposed method generates the neighbours using a kernel-like distance function such that no data preprocessing is needed. Moreover, unlike aLIME that uses a bottom-up construction procedure that favors high precision to discover pattern explanations, our method PALEX adopts contrast pattern mining techniques to generate contrast patterns as explanation candidates.

Importantly, most existing explanation systems only provide a single explanation for an individual prediction. In contrast, PALEX enriches explanations by providing multiple explanation candidates, which is useful in scenarios where multiple plausible explanations are reasonable. A comparison between PALEX and existing techniques is provided in Table 5.1.

Method	Multiple explanation support	Assumption that data is locally linearly separable	Representation of explanation	Categorical data support	Extra information requirement
Feature selection based (Robnik-Šikonja and Kononenko, 2008) (Henelius et al., 2014) (Adler et al., 2016) (Koh and Liang, 2017)	No	No	Vector of weights	Yes	No
Gradient-based (Baehrens et al., 2010) (Wang et al., 2016) (Selvaraju et al., 2017)	No	No	Vector of weights	No	No
LIME (Ribeiro, Singh, and Guestrin, 2016c)	No	Yes	Vector of weights	Yes	No
aLIME (Ribeiro, Singh, and Guestrin, 2018)	No	No	Pattern	Yes	No
PALEX (proposed)	Yes	No	Pattern	Yes	Frequent patterns from experts/training data

TABLE 5.1: Comparison of local explanation methods.

5.3 Patterns as Explanations

There are two common representations of explanations: *vectors of weights*, and *patterns*. When using vectors of weights as explanations, each weight corresponds to the predictive power of an attribute. A vector of weights is arguably more complex for a user to comprehend than a pattern, given the simpler format of patterns. Moreover, if there exist multiple explanations, a single vector of weights is unable to capture all the required information. Patterns also have the potential to provide meaningful value ranges for numeric attributes. Exploring meaningful value ranges is not considered in this chapter, but this direction is an interesting area for future work. Next, we give a formal definition of patterns as explanations (similar to the definition in (Martens and Provost, 2011)).

First, we introduce the notation $\mathcal{G}(f, x, p)$ that takes a classifier f , an instance x and a pattern p as inputs (assume $f(x)$ gives the class label c), and outputs the probability that the prediction is still $c = f(x)$ if p is violated for x , and is defined as:

$$\mathcal{G}(f, x, p) = \mathbb{E}_{x' \in x \setminus p} [\mathbb{P}_f(c|x')] \quad (5.1)$$

where $\mathbb{E}[\cdot]$ calculates the expectation, $\mathbb{P}_f(c|x)$ is the probability that x is classified as c by f and $x \setminus p$ denotes all possible instances where all conditions in p are violated for x .

Assume a dataset consists of two features $F_1, F_2 \in \{v_1, v_2, v_3\}$ and class label $c \in \{0, 1\}$. Given an instance $x : F_1 = v_1, F_2 = v_1$, a pattern $p : (F_1 = v_1)$, and a classifier f that give prediction $f(x) = 1$ with probability $\mathbb{P}_f(c = 1|x) = 0.8$, there are two possible perturbations of x to violate p such that $x \setminus p = \{x_1, x_2\}$ where $x_1 : F_1 = v_2, F_2 = v_1$ and $x_2 : F_1 = v_3, F_2 = v_1$. Suppose x_1, x_2 are equally likely distributed, and $\mathbb{P}_f(c = 1|x_1) = 0.4$, $\mathbb{P}_f(c = 1|x_2) = 0.3$, then $\mathcal{G}(f, x, p) = \mathbb{E}_{x' \in \{x_1, x_2\}} [\mathbb{P}_f(c = 1|x')] = \frac{1}{2}\mathbb{P}_f(c = 1|x_1) + \frac{1}{2}\mathbb{P}_f(c = 1|x_2) = 0.35$, meaning that the probability of being classified as $c = f(x)$ is 0.35 if x is violated by p .

Exact calculation of $\mathbb{E}_{x' \in x \setminus p} [\mathbb{P}_f(c|x')]$ is usually computationally infeasible (especially for numeric features when there are an infinite number of instances in $x \setminus p$), thus the approximation is computed by $\mathcal{G}(f, x, p) = \mathbb{E}_{x' \in x \setminus p} [\mathbb{P}_f(c|x')] \approx \frac{1}{M} \sum_{i=1}^M \mathbb{P}_f(c|z_i)$. Here z_i is a generated sample where feature values are the same as x for features that do not appear in pattern p , and feature values are randomly chosen to violate p for features that appear in p .

Next, we can define the notation of explanations using patterns. Given an instance x and a classifier f , a pattern p is called an explanation for $f(x) = c$ if:

1. x matches p ,
2. $\mathbb{P}_f(c|x) > \mathcal{G}(f, x, p)$.

The first property ensures that the explanation contains correct attribute values from x . The second property says that if a pattern is violated, the probability of being predicted as the same class label should drop. In other words, the explanation should be the supporting evidence for the prediction.

5.4 Proposed Algorithm: PALEX

We next describe our proposed method - Pattern Aided Local Explanations (PALEX). PALEX discovers a set of contrast patterns as the explanations for an individual prediction. In order to study the behavior of a given classifier in the vicinity of the test

instance being predicted, PALEX initially generates new neighbour samples for the instance. PALEX uses frequent patterns to guide the generation of these neighbours, according to the intuition that valid neighbours must contain matching frequent patterns in order to be close to the test instance. Closeness is measured using a distance function in pattern space. Then these generated instances are labeled by the given classifier: instances with the same label as the testing instance are treated as the positive set and the rest as the negative set. Contrast patterns for the positive set against the negative set are then mined as explanations, and these patterns capture the discriminative features for the testing instance against its neighbours with a different class label.

PALEX differs from LIME in two key aspects: (1) Unlike LIME which uses Euclidean distance, PALEX uses the input frequent patterns to generate a more precise neighbourhood for the given instance. (2) Instead of using an interpretable model to mimic the local behavior, PALEX applies a contrast pattern mining technique to capture locally important information. Comparing with aLIME (Ribeiro, Singh, and Guestrin, 2018), a variant of LIME, PALEX does not require successful detection of the local boundary, whereas for aLIME, a poorly detected boundary may result in degraded performance. aLIME employs a bottom-up strategy to identify the explanation pattern and the construction process is targeted towards high precision scenarios. PALEX selects the explanation pattern(s) from a set of easily mined candidates (contrast patterns) and is targeted towards optimising both precision and recall.

The number of explanations found by contrast pattern mining could be very large, and needs to be pruned. An additional step to extract the top K representative explanations is carried out in the final phase. We formulate this process as a linear optimization problem with multiple constraints.

The steps of PALEX are shown in Algorithm 6, and graphically illustrated in Figure 5.1.

Algorithm 6: PALEX

Input: classifier f , instance x , frequent pattern set PS , number of samples N , minimal support $minSupp$, minimal growth ratio $minRatio$, number of explanations K

Output: K contrast patterns as explanations

```

1  $nei = generateNeighbours(x, PS, N);$ 
2 for  $z$  in  $nei$  do
3    $z.label = f(z)$ 
4 end
5  $cp = mineCPs(nei, minSupp, minRatio, x.label);$ 
6 remove contrast patterns in  $cp$  that do not satisfy the explanation definition
   properties in Section 5.3;
7  $explanations = selectExplanations(cp, K, f)$  (Solve Equation 5.7);
8 return  $explanations$ ;
```

5.4.1 Methodology

Input

The inputs for PALEX include: (1) **classifier** f , which can be any given classifier and it is treated as a black box. The explanation process requires it to provide probabilities (or similar scores) for the predictions; (2) **instance** x , which is the instance whose prediction by f needs to be explained; (3) **frequent pattern set** PS , which can be considered as a

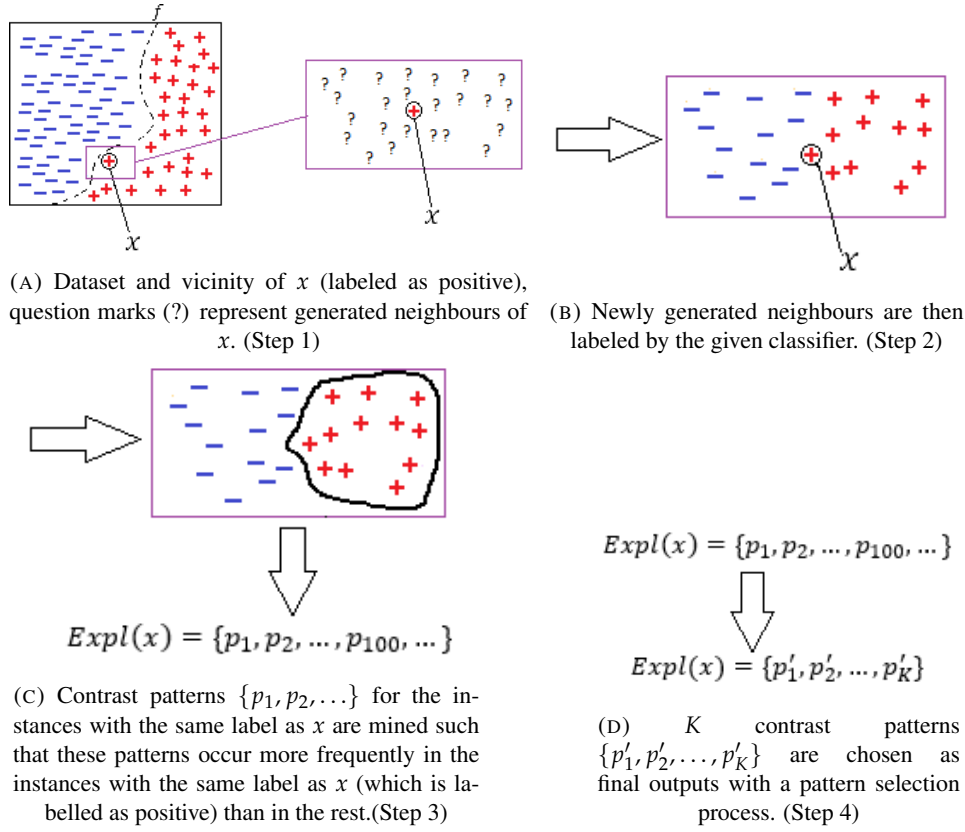


FIGURE 5.1: Overview of PALEX process

summary or compression of the data. It can be mined from the data used to train f , or provided by the users based on their domain expertise; (4) **number of samples** N , which is the number of neighbours generated in the vicinity of x . A large value of N can slow down the contrast pattern mining step. On the other hand, a small value of N is not sufficient to simulate the behavior of f in the vicinity of x ; (5) **minimal support** $minSupp$ and (6) **minimal growth ratio** $minRatio$ are the thresholds used to mine contrast patterns. If these two values are set too large, the number of contrast patterns mined may be too small and thus useful explanations could be ignored. On the other hand, if they are set too small, there will be numerous contrast patterns, which makes it harder to choose representatives; (7) **number of explanations** K , which is the threshold for the number of explanations such that at most K explanations are returned.

Output

K contrast patterns satisfying the definition (Section 5.3) are returned.

Process

The process of PALEX is as follows.

First, N neighbours of x are generated (described in Algorithm 6). PALEX perturbs x to generate new instances by random perturbation such that at each generation for each feature it tosses a coin to decide whether it should be perturbed or not, and if yes, assigns a possible value (which can be any item for a categorical feature, or any value in the domain range for a numerical feature) to the feature. Each newly generated instance z is weighted

Algorithm 7: generateNeighbours

Input: instance x , frequent pattern set PS , number of samples N
Output: N instances

```

1  $nei = \{\}$ ;
2 while  $nei.size < N$  do
3    $z = x.copy()$ ;
4   uniformly select a subset of features  $z$  and uniformly assign a possible value
     to each chosen feature.
5    $z.weight = \exp^{-(Dist(x,z|PS))^2}$  (where  $Dist(x,z|PS)$  is defined in Equations
     5.2 and 5.3);
6    $nei = nei \cup \{z\}$ 
7 end
8 return  $nei$ ;
```

by $\exp^{-(Dist(x,z|PS))^2}$, where $Dist(x,z|PS)$ (see Equations 5.2 and 5.3) is the distance (in frequent pattern space based on PS) of z to x .

Second, the class labels of the newly generated samples are predicted by f .

Third, contrast patterns that occur frequently in the samples with the same class label as x and infrequently in other classes are extracted. The pattern mining algorithm should be able to handle weighted instances, as the neighbours are weighted by their distances to the test instance. For example, a common strategy is to calculate the weighted support by $supp(p, D) = \frac{\sum_{x_j \in mds(p, D)} x_j.weight}{\sum_{x_i \in D} x_i.weight}$, where $mds(p, D)$ is the matching dataset of p in D . For example, given a dataset $D = \{x_1, x_2, x_3\}$ where the instances are weighted by $x_1.weight = 0.1, x_2.weight = 0.2, x_3.weight = 0.3$, and a pattern p that matches x_1 and x_2 , the weighted support of p is $supp(p, D) = \frac{x_1.weight + x_2.weight}{x_1.weight + x_2.weight + x_3.weight} = 0.5$. Most contrast pattern mining methods that are able to deal with instance weights can generate good explanation candidates. For efficiency purposes, we use a random forest based mining process that was proposed in (Shang et al., 2016) (similar to (Kang and Ramamohanarao, 2014)), and one can also replace this with other mining methods that have similar performance with the same thresholds. For other contrast pattern mining algorithms, interested readers can refer to (Dong and Bailey, 2012).

Lastly, K patterns are selected from the set of contrast patterns. Patterns that do not meet the definition of an explanation are removed, then at most K contrast patterns are selected with an optimization step (outlined in Section 5.4.3) from the remaining set.

5.4.2 Sampling in the frequent pattern space

We assign weights to the newly generated instances using a distance function that measures the distances of these instances to the given test instance, such that a closer instance is assigned a higher weight, making it more influential in the explanation process that follows.

Given a frequent pattern set $PS = \{p_1, p_2, \dots, p_M\}$, an instance x is mapped to an M -dimensional numeric vector x' , where the i -th feature is non-zero if x matches p_i , representing the strength of p_i (support is used in this chapter). The distance between two instances x, z is measured by $D(x', z')$, where x', z' are their projections in the frequent pattern space. $D(x', z')$ can be any traditional distance measure and l_1 distance is used in this chapter. Formally,

$$Dist(x, z|PS) = D(x', z') = \sum_i |x'_i - z'_i| \quad (5.2)$$

where $x' = (x'_1, \dots, x'_M)$ such that

$$x'_i = \phi(x_i) = \begin{cases} \text{supp}(p_i), & x \text{ matches } p_i \\ 0, & \text{otherwise} \end{cases} \quad (5.3)$$

and z' is defined similarly.

This technique with a feature map $\Phi(x) = (\phi(x_1), \dots, \phi(x_M))$ is similar to a kernel method that maps the original features into a certain feature space. Frequent patterns mined from the training data (or obtained from experts with background knowledge) contain information about important correlations. When generating a neighbourhood (vicinity) around the instance, it is important to preserve correlations (frequent patterns) possessed by the test instance. Intuitively speaking, a frequent pattern that is shared between the test instance and its neighbours acts as a connection between them. The more connections two instances have, conceptually the closer they are (like in a social network the more mutual friends two people have, the higher the probability that they are friends). Measuring distance in the frequent pattern space calculates the connections (shared correlations) between instances.

Sampling in the frequent pattern space can be treated as an example of sampling in the local embedding (see Chapter 6) if the patterns are generated from the vicinity of an instance, and the coordinates of the local space consists of the patterns.

5.4.3 Explanation selection

This process requires the classifier f to be able to provide probabilities or similar confidence scores for its predictions in order to evaluate the power of a contrast pattern as an explanation. The scoring function of contrast pattern p is defined as

$$\text{score}(p|x, f) = \mathbb{P}_f(c|x) - \mathcal{G}(f, x, p) = \mathbb{P}_f(c|x) - \mathbb{E}_{x' \in x \setminus p}[\mathbb{P}_f(c|x')] \quad (5.4)$$

where $c = f(x)$ is the prediction of x by f , and $\mathbb{P}_f(c|x)$ is the corresponding probability. $\mathcal{G}(f, x, p)$ is the probability that the prediction is still c if p is violated (see Equation 5.1). Note that the scoring function is different from the support or growth ratio of a pattern, as the scoring function measures the prediction probability/confidence change if the pattern is violated, while the support or growth ratio is an intrinsic property of the pattern in the data.

Given contrast patterns CP mined from Step 3, the choice of K of them as explanations is formulated as the following optimization problem:

$$\begin{aligned} \text{expl}(x) &= \underset{P \subseteq CP}{\operatorname{argmax}} \{L(f, P) - \alpha \Omega(P)\} \\ &\text{subject to } |P| \leq K \end{aligned} \quad (5.5)$$

where $L(f, P) = \sum_{p \in P} \text{supp}(p) * \text{score}(p|x, f)$ which is a weighted summation of the pattern score, α is the regularization rate (we set it to 1 as the experiments suggest that it achieves best performance on average when multiple explanations are required) and $\Omega(P)$ computes the average overlap of pair-wise patterns as the penalty term. We use the Jaccard similarity, which is defined as

$$\Omega(P) = \sum_{p_i, p_j \in P, i \neq j} \frac{|\text{mds}(p_i) \cap \text{mds}(p_j)|}{|\text{mds}(p_i) \cup \text{mds}(p_j)|}. \quad (5.6)$$

Other overlap measuring methods can also be used. The overlap constraint forces the selected patterns to be as diverse as possible.

This optimization problem can be re-formulated as a MIP (Mixed Integer Programming) problem and solved by a stochastic MIP solver. It is re-formulated as:

$$\begin{aligned}
& \max_{a_i, b_{ij}} \quad a_i * score_i - b_{ij} * penalty_{ij} \\
& \text{subject to} \quad \sum_i a_i \leq K \\
& \quad \quad \quad a_i + a_j - 2b_{ij} \leq 1, \forall i, j \\
& \quad \quad \quad a_i, b_{ij} \in \{0, 1\}
\end{aligned} \tag{5.7}$$

where a_i, b_{ij} are the decision variables, $a_i = 1$ if pattern p_i is chosen and $b_{ij} = 1$ if patterns p_i, p_j are chosen, $score_i = score(p_i|x, f)$ and $penalty_{ij} = \frac{|mds(p_i) \cap mds(p_j)|}{|mds(p_i) \cup mds(p_j)|}$. The first constraint makes sure that at most K patterns are selected. The second constraint defines the relation of a_i and b_{ij} , and the last constraint defines the type of the decision variables. Users can also add other optional constraints of their own choice. We have tried constraints in terms of pattern length, support and ratio, but there is no significant improvement.

In particular, if $K = 1$ and only one explanation is required, the problem is transformed to find the explanation with the highest weighted score in Equation 5.4.

5.4.4 Example

The example is adopted from the UCI balloon data with a custom classification rule. The dataset D consists of four features: color, size, act, age that are denoted as x_1, x_2, x_3, x_4 respectively, such that x_1 (color) $\in \{\text{YELLOW}, \text{PURPLE}\}$, x_2 (size) $\in \{\text{SMALL}, \text{LARGE}\}$, x_3 (act) $\in \{\text{STRETCH}, \text{DIP}\}$, x_4 (age) $\in \{\text{ADULT}, \text{CHILD}\}$, and the class label $c \in \{T, F\}$. Assume the underlying classification model f is:

$$f(x) = \begin{cases} T, & (x_1=\text{YELLOW} \wedge x_2=\text{SMALL}) \vee (x_3=\text{STRETCH} \wedge x_4=\text{ADULT}) \\ F, & \text{otherwise} \end{cases}$$

Two test instances x_1 and x_2 are shown in Table 5.2. Three methods PALEX, LIME and aLIME are used to generate explanations for the predictions of x_1, x_2 by the underlying model f while treating f as a black box.

	color	size	act	age
x_1	YELLOW	SMALL	STRETCH	CHILD
x_2	YELLOW	SMALL	STRETCH	ADULT

TABLE 5.2: Test instances to illustrate explanations.

Single explanation case

Instance x_1 is used to test the single explanation scenario. The results are shown in Figure 6.1c. For instance x_1 , it can be inferred from the model f that the true explanation is “because color is YELLOW and size is SMALL”. LIME gives a vector of weights representing the contributions/importance of each feature, and it gives higher importance to features *act* and *age* which are not precise. aLIME gives an explanation that includes

Method	Explanation
LIME	(0.29, 0.28, 0.51, -0.47)
aLIME	{size = SMALL, act = STRETCH}
PALEX	{color = YELLOW, size = SMALL}

(A) Single explanation case

Method	Explanation
LIME	(0.29, 0.26, 0.52, 0.49)
aLIME	{size = SMALL, act = STRETCH}
PALEX	{color = YELLOW, size = SMALL} OR {act = STRETCH, age = ADULT}

(B) Multiple explanations case

FIGURE 5.2: Comparison of explanations generated by LIME, aLIME and PALEX.

feature *act* but ignores feature *color*. PALEX provides an explanation that matches the true explanation.

Multiple explanations case

One advantage of PALEX is its ability to generate multiple explanations. Not only does it enable users to choose the best explanations from a small number of candidates, but it also naturally fits the scenario where there exist multiple true explanations. Instance x_2 is used for testing the multiple explanations scenario, the results are shown in Figure 6.1d. Given the classification rule and the instance, it is clear that the two possible explanations are “because color is YELLOW and size is SMALL” or “act is STRETCH and age is ADULT”. LIME gives a single explanation suggesting all features are important, but fails to capture the underlying truth, while aLIME gives a single explanation saying “because color is YELLOW and act is STRETCH”. In contrast, PALEX generates both of the two explanations.

Multiple explanations can be useful in the scenario where the users want to generate exploratory insights from a model. Each explanation/insight can reveal some logical reasoning ideas from the model, and it is left to the users to verify whether it holds in reality or not.

5.4.5 Complexity analysis

PALEX consists of three major steps: synthetic neighbour generation, contrast pattern mining and pattern selection. Neighbour generation iteratively computes distances in the pattern space for N neighbours, and its complexity is $O(NL)$, where $L = |PS|$ is the size of the pattern set and will be influenced by data dimensionality if the pattern set is mined directly from the data. For contrast pattern mining, it is desirable to employ an efficient mining process. We use the random forest based mining method whose complexity is $O(TNd)$, where T is the number of trees and d is the maximum tree depth. For the pattern selection process, solvers normally take polynomial time in terms of the number of variables and constraints, thus the complexity is $O(S^{2c})$, where S is the number of contrast patterns generated by the previous step, and c is some positive number (e.g., 2.5 for Karmarkar’s algorithm (Karmarkar, 1984)). Overall, the total complexity is $O(N(L + Td) + S^{2c})$. An important challenge in scaling for large datasets is that a

frequent pattern set summarizing the training data is required as input, but since this mining process happens only once, existing efficient pattern mining methods for big data (see (Aggarwal and Han, 2014)) can be applied. Moreover, this problem can be avoided if the frequent patterns come from expert knowledge.

5.5 Experiments

In this section, we evaluate the quality of explanations in order to answer the question of whether the explanations are faithful to the model. The core idea of measuring the faithfulness of the explanations is to compare them with the true ones that are completely faithful to the model. In our experiments, the true explanations are either pre-defined using known generation of synthetic data, or extracted by looking inside an interpretable model (decision trees are used in this chapter). The impact of the explanation selection process is also discussed.

5.5.1 Metrics

We adopt the explanation evaluation method of Eq. 2.2 in Chapter 2. Given a true explanation e_{true} , a D -dimensional instance x and a model f whose explanation for x is e , the faithfulness of the explanation is measured by the F_1 score:

$$F_1 = 2 \frac{precision * recall}{precision + recall} \quad (5.8)$$

where $precision(e, e_{true}) = \frac{\sum_{d \in D} |e_d \times e_{true,d}|}{\sum_{d \in D} |e_d|}$ represents the fraction of features occurring in the explanation that are true, and $recall(e, e_{true}) = \frac{\sum_{d \in D} |e_d \times e_{true,d}|}{\sum_{d \in D} |e_{true,d}|}$ represents the fraction of true features that the explanation covers.

5.5.2 Experimental setup

The experiments are conducted on four synthetic datasets and ten UCI datasets. For the synthetic data, the oracle classifiers and true explanations are predefined. For UCI data, decision trees are built on training data and the true explanations are extracted from the structures (prediction paths) of the trees. The averaged metrics over the testing data are calculated.

Benchmarks. Our method PALEX is compared with *LIME*, a local explanation system that computes the contributions of features for an individual prediction, *aLIME* (a variation of LIME), and a global explanation method (denoted as *Global*). For LIME, a set of features with values that are in favor of the prediction and with a contribution (weight) greater than 0.01 is extracted to form an explanation pattern (contrast pattern) that supports the prediction. For aLIME, the anchor rule generated is used as the explanation. The parameters used for LIME and aLIME are chosen via a validation process (similar to the process described later for PALEX) for optimal performance. For Global (similar to (Koh and Liang, 2017)), the training data is relabeled by the given classifier, and a set of classification rules are generated, then the matching rule is selected as the explanation.

Parameter settings In our experiments, for PALEX, the initial frequent pattern set is obtained from the training data via FP-growth (Han, Pei, and Yin, 2000) with minimal support = 0.1. Parameter optimization via a nested validation process during the training process is a common way for classification tasks, however, it is difficult for explanation systems due to the fact that true model explanations are always unavailable

when deploying such systems in real-life problems. A solution to decide the parameters is to generate a parameter setting using validation datasets whose true model explanations are available, and then apply the same setting for any new dataset. To simulate such a process, we apply PALEX with different parameter settings of three major parameters N , $minSupp$ and $minRatio$ on four separate UCI datasets (*adult*, *crx*, *hepatitis* and *ILPD*, which are different from those in Table 8.3 used for evaluation) as validation datasets, then the recommended parameter setting (that achieves the best F_1 score on average over the validation datasets) is used for all the datasets whose results are reported for comparison and analysis in this section. More specifically, each validation dataset is randomly split into two groups such that 80% is for training as T_1 and 20% is for test as T_2 . Then grid search is used to find the optimal parameter setting. Every possible combination $(N, minSupp, minRatio)$ where $N \in \{50, 200, 500, 1000, 3000, 5000\}$, $minSupp \in \{0.1, 0.15, 0.2, 0.3, 0.4\}$ and $minRatio \in \{0.1, 0.15, 0.2, 0.3, 0.4\}$ (the possible values are commonly used in pattern based methods) is tried on T_1 and evaluated on T_2 for all four validation datasets. There are 150 possible combinations, and the combination $(N^*, minSupp^*, minRatio^*)$ with best performance on average is chosen as the parameter setting for the rest of our experiments. The parameter setting that we find is $N^* = 500, minSupp^* = 0.1, minRatio^* = 5$. Similarly, the parameters for the benchmark methods are obtained via the same validation datasets, and a fixed setting is used for all other test data.

For the choice of the number of explanations K in PALEX, unless otherwise specified, we set it to 1 for evaluation purposes as all baselines generate only one explanation.

When deploying the PALEX system in real-life problems, users can employ their own validation dataset (i.e., one can use explanations that come from domain experts to approximate true model explanations) to repeat the process of generating parameter settings.

5.5.3 Faithfulness to oracle on synthetic data

We first conduct experiments on synthetic data. The advantage of using synthetic data is that we can test the explanation systems with an oracle classifier and easily obtain the true explanations. Four synthetic datasets are generated using the following DNFs:

1. $(f_1 \wedge f_2 \wedge f_3) \vee (\neg f_1 \wedge f_4 \wedge f_5)$, where $f_1, \dots, f_5$ are binary attributes.
2. $((f_1 == 1) \wedge f_2 \wedge f_3) \vee ((f_1 == 2) \wedge f_4 \wedge f_5) \vee ((f_1 == 3) \wedge f_6 \wedge f_7) \vee ((f_1 == 4) \wedge f_8 \wedge f_9)$, where f_1 is numeric and $f_2, \dots, f_9$ are binary attributes.
3. $(f_1 \wedge f_2) \vee (f_3 \wedge f_4) \vee (f_5 \wedge f_6) \vee (f_7 \wedge f_8)$, where $f_1, \dots, f_8$ are binary attributes.
4. $(f_1 \wedge f_2 \wedge f_3) \vee (f_4 \wedge f_5 \wedge f_6) \vee (f_7 \wedge f_8 \wedge f_9)$, where $f_1, \dots, f_9$ are binary attributes.

For each generation of the synthetic datasets, the feature values are randomly chosen and the class label is $T(true)$ if the corresponding DNF is satisfied. As the DNFs are used to construct the oracle classifier, no training data is required. Since the negation of a DNF rule is also a DNF, it is sufficient to examine the explanations for instances with label T .

Synthetic-1 and Synthetic-2 are used to test the *single-explanation* case and the other two datasets are used for the *multiple-explanation* case (where the number of generated explanations K is set to 2). The true explanation for an instance with label T is the set of features that occur in the clause satisfied by the instance. For example, assume the rule is $(f_1 \wedge f_2) \vee (f_3 \wedge f_4)$, for instance $(f_1 = true, f_2 = true, f_3 = false, f_4 = true)$, the satisfying clause is $(f_1 \wedge f_2)$, thus the true explanation is $\{f_1 = true, f_2 = true\}$.

Similarly, for instance $(f_1 = \text{true}, f_2 = \text{true}, f_3 = \text{true}, f_4 = \text{true})$, there are two possible explanations $\{f_1 = \text{true}, f_2 = \text{true}\}$ or $\{f_3 = \text{true}, f_4 = \text{true}\}$.

Results and discussion. For the multiple-explanation case, results are the average value of all the possible explanations. The results are reported in Table 5.3 and Table 5.4, and the results show that PALEX is more faithful to the oracle classifier.

Dataset	Precision				Recall			
	PALEX	LIME	aLIME	Global	PALEX	LIME	aLIME	Global
Syn-1	1.00	0.44	0.89	0.65	1.00	1.00	0.65	0.87
Syn-2	0.99	0.50	0.48	0.44	0.82	0.74	0.27	0.77
Syn-3	0.87	0.48	0.83	0.65	0.94	1.00	0.83	0.88
Syn-4	1.00	0.54	1.00	0.61	0.76	0.80	0.47	0.69

TABLE 5.3: Precision and recall results for synthetic data.

Dataset	F_1 (faithfulness)			
	PALEX	LIME	aLIME	Global
Syn-1	1.00	0.68	0.74	0.72
Syn-2	0.86	0.59	0.34	0.64
Syn-3	0.90	0.64	0.83	0.76
Syn-4	0.83	0.70	0.62	0.63

TABLE 5.4: F_1 (faithfulness) results for synthetic data.

5.5.4 Faithfulness to decision trees on UCI data

We conduct experiments on a variety of UCI datasets. The true explanations are extracted in the same way as LIME: decision trees are trained from the training data, and the true explanations for the test data are obtained from the trees, though the explanation systems treat the decision trees as black boxes.

Data. The datasets used are described in Table 8.3. Each dataset is randomly split into two groups such that 80% is used to train the classifier and the remaining 20% is used to evaluate the explanations.

Dataset	#inst	#nomAttr	#numAttr	#classes
Balloon	16	4	0	2
Blood	758	0	4	2
Breast-cancer	596	8	0	2
Diabetes	768	0	8	2
Ionosphere	351	0	34	2
Iris	150	0	4	3
Labor	57	8	8	2
Musk	6598	0	168	2
Titanic	3772	22	7	2
Vote	435	16	0	2

#inst - number of instances, #nomAttr - number of nominal attributes, #numAttr - number of numeric attributes, #classes - number of classes

TABLE 5.5: UCI datasets description.

Classifiers and true explanations. Decision trees are trained for all datasets. The true explanation for the prediction of an instance is the set of features that occur in the path from the root to the leaf corresponding to the prediction.

Results and discussion. The metrics averaged over all test instances for each dataset are reported in Table 5.6 and Table 5.7. For precision, PALEX is able to achieve the best results in 7 out of 10 datasets. For recall scores, PALEX still wins in 6 out of 10 datasets and achieves comparable results with LIME and aLIME in the others. In terms of the F_1 score, PALEX wins in 8 out of 10 datasets. It also can be seen that PALEX achieves stronger precision than recall when there is only one explanation that is generated, which is desired as the explanation is the selected most representative pattern from many candidates. When increasing K - the number of explanations, it can be expected that the recall would increase and precision may drop as more features may occur in the explanations.

5.5.5 Impact of sampling method

Gradient based sampling techniques have been widely used in the field of adversarial classification, where nearby instances with the opposite class label to a given instance are generated. We compare a gradient based sampling method with the proposed pattern based one. In the gradient based sampling method, the objective function is defined as $g(x') = \text{dist}(x, x') + L(\text{Pr}(y|x), \text{Pr}(y|x'))$, where $\text{dist}(x, x')$ is the cost of modifying x to x' (l_2 distance is used) and $L(\text{Pr}(y|x), \text{Pr}(y|x'))$ measures the prediction probability difference (l_1 distance is used). To find the instance of the opposite class with minimal cost, the given instance x is moved towards the direction $-\lambda \Delta g(x')$ where λ is the step size. Because the gradient based sampling method always generates samples along one direction towards a local optima, we run the methods several times until the desired number of samples are found and add a constraint that new generated instances should be far enough to instances that are found in the previous runs. The desired number of samples is set to 500 in this experiment. We also report the results of a random generation method, which randomly selects a feature, uniformly assigns a new feature value and weights it using the distance from the test instance at each generation of new instance, as used in LIME. In this experiment, since gradient-based methods are applicable only on numeric datasets, a process of conversion from nominal features to numeric features is conducted beforehand. The experimental results (Table 5.8) show that gradient based sampling is not as competitive as the proposed method, especially for high-dimensional data. One possible reason is that given the same size limit on the sample set, the instances generated by the gradient method are still not sufficiently diversified. Another major drawback of the gradient based method is that it is difficult to apply to datasets with nominal features.

5.5.6 Impact of explanation selection

In this part, we investigate the impact of explanation selection (the last step of PALEX) on the faithfulness of PALEX. The explanation selection process is a key step in the proposed method. The goal of this step is to reduce the number of explanation candidates and preserve the quality. The number of explanations K is set to 5. We compare the performance (best F_1 score) of the explanations *before* and *after* pruning in Table 5.9. It can be seen that the performance of the explanations before pruning is an upper bound for those after pruning, because the explanations of the latter are a subset of the former. The numbers of explanations (#Expl) show that PALEX before pruning generates more than 50 explanations for datasets with more than four features, and the selection procedure consistently extracts less than four explanations, which suggests that the output remains the same for any choice of threshold K that is greater than 4. In terms of the quality, the

Dataset	Precision				Recall			
	PALEX	aLIME	LIME	Global	PALEX	aLIME	LIME	Global
Balloon	1.00(0.00)	1.00(0.00)	0.5(0.00)	0.5(0.00)	0.88(0.22)	0.88(0.22)	1.00(0.00)	0.75(0.18)
Blood	0.97(0.12)	0.88(0.33)	0.78(0.25)	0.47(0.14)	0.72(0.28)	0.67(0.35)	0.78(0.24)	0.72(0.21)
Breast-cancer	0.96(0.18)	1.00(0.00)	0.61(0.26)	0.58(22)	0.92(0.23)	0.92(0.18)	0.90(0.25)	0.88(0.14)
Diabetes	0.98(0.08)	0.97(0.13)	0.79(0.27)	0.67(0.20)	0.82(0.19)	0.70(0.22)	0.58(0.28)	0.78(0.16)
Iris	1.00(0.00)	1.00(0.00)	0.83(0.37)	0.64(0.28)	1.00(0.00)	1.00(0.00)	0.83(0.37)	0.89(0.14)
Ionosphere	0.90(0.21)	0.83(0.29)	0.21(0.16)	0.44(0.25)	0.57(0.24)	0.55(0.26)	0.33(0.34)	0.79(0.32)
Labor	0.96(0.14)	1.00(0.00)	0.85(0.23)	0.68(0.15)	0.97(0.09)	0.97(0.10)	0.73(0.09)	0.77(0.06)
Musk	0.91(0.12)	0.84(0.18)	0.79(0.14)	0.56(0.15)	0.81(0.18)	0.64(0.22)	0.75(0.11)	0.68(0.33)
Titanic	0.78(0.18)	0.96(0.16)	0.62(0.28)	0.58(0.15)	0.82(0.22)	0.64(0.26)	0.50(0.24)	0.69(0.21)
Vote	0.99(0.08)	0.74(0.32)	0.83(0.26)	0.55(0.24)	0.92(0.17)	0.69(0.22)	0.56(0.24)	0.62(0.17)

TABLE 5.6: Precision and recall results of PALEX, aLIME, LIME and Global on UCI datasets (the best results are highlighted in bold, and standard deviations are included in parentheses)

Dataset	F_1 (faithfulness)			
	PALEX	aLIME	LIME	Global
Balloon	0.92(0.14)	0.92(0.14)	0.67(0.00)	0.59(0.12)
Blood	0.79(0.20)	0.74(0.32)	0.72(0.14)	0.52(0.20)
Breast-cancer	0.94(0.20)	0.92(0.12)	0.70(0.22)	0.77(0.22)
Diabetes	0.85(0.12)	0.79(0.17)	0.63(0.23)	0.68(0.24)
Iris	1.00(0.00)	1.00(0.00)	0.83(0.37)	0.74(0.24)
Ionosphere	0.63(0.15)	0.65(0.27)	0.22(0.14)	0.54(0.28)
Labor	0.96(0.10)	0.98(0.05)	0.88(0.16)	0.72(0.09)
Musk	0.86(0.12)	0.72(0.21)	0.76(0.15)	0.64(0.31)
Titanic	0.77(0.13)	0.73(0.21)	0.52(0.23)	0.61(0.19)
Vote	0.94(0.12)	0.70(0.26)	0.84(0.23)	0.59(0.11)

TABLE 5.7: F_1 (faithfulness) results of PALEX, aLIME, LIME and Global on UCI datasets (the best results are highlighted in bold, and standard deviations are included in parentheses)

Dataset	Pattern-based	Gradient-based	Random
Balloon	0.92	0.88	1.00
Blood	0.79	0.69	0.72
Breast-cancer	0.94	0.72	0.68
Diabetes	0.85	0.42	0.73
Iris	1.00	0.56	0.85
Ionosphere	0.63	0.35	0.49
Labor	0.96	0.65	0.59
Musk	0.86	0.64	0.56
Titanic	0.77	0.26	0.65
Vote	0.94	0.79	0.65

TABLE 5.8: Faithfulness (F_1 score) of different sampling methods on numeric data

explanations after pruning achieve comparable metrics in most datasets. The running time (in seconds) is also reported. The running environment is on Windows 10 with processor Intel(R) Core(TM) i7-5500U CPU 2.40GHz and 8.00 GB RAM.

5.5.7 Impact of parameters

We also conducted experiments on a range of UCI datasets (blood, diabetes, ionosphere, titanic and vote) to investigate the impact of the parameter settings used by PALEX, and the results are shown in Figure 5.3. The performance (F_1 score) increases as the sample size N increases, then becomes stable after $N \geq 500$. A large sample size may slow down the running time as it has an impact on the later contrast pattern mining process (the higher the sample size is, the slower the mining process will be), and the experiments show that $N = 500$ is a reasonable setting as it is the minimum threshold that achieves good performance on average. The performance increases as $minSupp$ increases, and then it reaches a maximum and decreases after $minSupp$ is greater than a specific value (which is usually between 0.1 to 0.2, and depends on the specific dataset). PALEX achieves the best metrics on average when $minSupp = 0.1$. The performance decreases as $minRatio$ increases and achieves the best results on average when $minRatio = 2$. As in the case of high $minSupp$ and high $minRatio$ the number of patterns reduces substantially, hence it misses many valid explanations and causes degradation in the the performance as can be seen in Figure 5.3b and Figure 5.3c.

At last, we investigate the impact of α (regularization rate in the explanation selection process), and K is set to 5 (K is required to be larger than 1 to enable the influence of α). A small α would generate explanations that are similar to each other, thus misses some explanations of good quality, and a large α pushes the explanations to be distinct to each other while significantly ignoring their quality. The performance (best F_1 score of K explanations) is plotted in Figure 5.3d and it achieves best performance on average when $\alpha = 1$.

5.6 Limitations and Future Work

We have shown the effectiveness and usefulness of PALEX and compared it to the benchmark methods, and now we turn to its limitations and possible directions for future work.

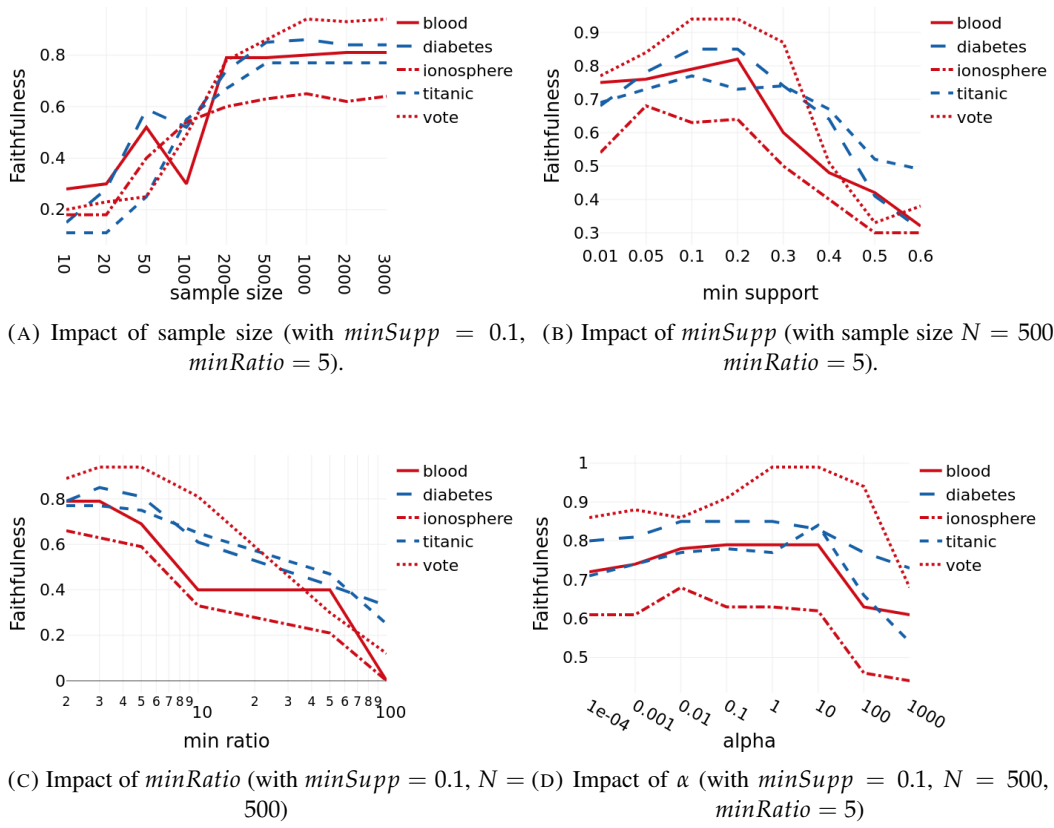
One limitation is that PALEX assumes the availability of a set of frequent patterns that summarizes the possible local classification regions/subspaces of the training data. If the pattern set is not of good quality (e.g., it cannot cover the whole sample space, or there is too much overlap), then the generation of neighbours will be degraded. It would be an interesting direction to push the frequent pattern mining phase within PALEX, to provide control over what type of frequent patterns are generated, with a view to optimisation of the downstream explanation task.

Another limitation is with respect to the evaluation challenges. The most common evaluation strategy is to evaluate the explanations extracted by the explanation extraction methods with a white-box model (from which the true model explanation can be inferred), and the model is treated as black-box for these methods. When evaluating with a real black-box model, the true explanations are never be available. One alternative might be to collect the true explanations from experts with background knowledge, but the issue is that there is often a gap between the model and the experts.

We also hope that local explanation methods encourage the collection of explanations while collecting the data. For example, the explanations could come from doctor reports in the medical diagnosis area, or from user reviews in a recommendation system. With

Dataset	Precision		Recall		F_1		#Expl		Running time (s)	
	before	after	before	after	before	after	before	after	before	after
Balloon	1.00	1.00	0.88	0.88	0.92	0.92	3.52	1.0	0.81	3.13
Blood	1.00	0.97	0.85	0.72	0.90	0.79	6.3	2.4	0.15	1.33
Breast-cancer	0.96	0.96	0.92	0.92	0.94	0.94	157.3	2.0	0.17	0.28
Diabetes	1.00	0.98	0.97	0.82	0.96	0.85	76.1	2.4	0.64	1.03
Iris	1.00	1.00	1.00	1.00	1.00	1.00	8.0	3.1	0.08	0.22
Ionosphere	0.93	0.90	0.86	0.57	0.81	0.63	244.6	2.6	1.40	3.78
Labor	1.00	0.96	1.00	0.97	1.00	0.96	526.5	2.0	0.23	0.51
Musk	0.91	0.91	0.83	0.81	0.88	0.86	725.1	3.8	30.54	58.00
Titanic	0.89	0.78	0.96	0.82	0.91	0.77	53.4	2.3	0.43	0.72
Vote	1.00	0.99	1.00	0.92	1.00	0.94	164.1	1.6	0.20	0.34

TABLE 5.9: Impact of selection process. #Expl denotes the number of explanations, “before” denotes the metrics before the pruning process and “after” denotes the metrics after the pruning process.

FIGURE 5.3: Impact of parameters on faithfulness (F_1 score)

such additional information, we may be able to improve a classifier to make more reliable predictions whilst the accuracy can be preserved.

5.7 Conclusion

In this chapter we proposed PALEX, a pattern aided approach to provide explanations for individual predictions. PALEX exploits the intrinsic interpretability of contrast patterns and uses them as a form of explanation and generates the vicinity of the instance being predicted using frequent pattern spaces. An explanation selection process is also proposed to prune large candidate sets. Experimental results show that PALEX is more faithful to models compared to benchmark methods and the selection process is able to effectively choose K representative explanations.

Chapter 6

Improving the Quality of Explanations with Local Embedding Perturbations

6.1 Introduction

Local explanations evaluate the behavior of a classifier in the vicinity of a given instance. A key step in this approach is to generate synthetic neighbors of the given instance. This neighbor generation process is challenging and it has considerable impact on the quality of explanations. To assess quality of generated neighborhoods, we propose a local intrinsic dimensionality (LID) based locality constraint. Based on this, we then propose a new neighborhood generation method. Our method first fits a local embedding/subspace around a given instance using the LID of the test instance as the target dimensionality, then generates neighbors in the local embedding and projects them back to the original space. Experimental results show that our method generates more realistic neighborhoods and consequently better explanations. It can be used in combination with existing local explanation algorithms.

When evaluating an explanation for the prediction of a black-box model, the quality of the explanation refers to the degree to which the explanation reflects how a model truly ‘thinks’. If an instance is difficult to explain, an extracted explanation with poor quality is not useful, as it fails to provide insights about the the logical reasoning of the model. An important question arises: *why is a particular instance difficult to explain?* We aim to improve the quality of explanations by addressing this question.

A general framework to obtain local explanations from a black-box model is to first generate a set of synthetic neighbors of a given instance, and then extract an explanation from these neighbors after they are labeled by the model. As observed in Chapter 5 and the work in (Laugel et al., 2018), *the quality of an explanation for a prediction depends heavily on the quality of the generated neighbor instances of the instance*. Despite the importance of the quality of the neighborhood generation, few studies have been conducted to improve the quality of explanations by generating neighbor instances of good quality (Fong and Vedaldi, 2017). Existing local explanation methods generally employ random perturbation and use Euclidean distance to threshold/weight the newly generated instances.

From another perspective, extracting the local explanation from a given test instance is similar to identifying the local classification boundary of a given model. Suppose that the local classification boundary lies within a local subspace around the test instance, we can categorize synthetic neighbor instances into ‘good’ neighbors and ‘bad’ ones. We could build a local model that is similar to the underlying true model from ‘good’ synthetic neighbors, whereas a local model trained on the ‘bad’ synthetic neighbors is far

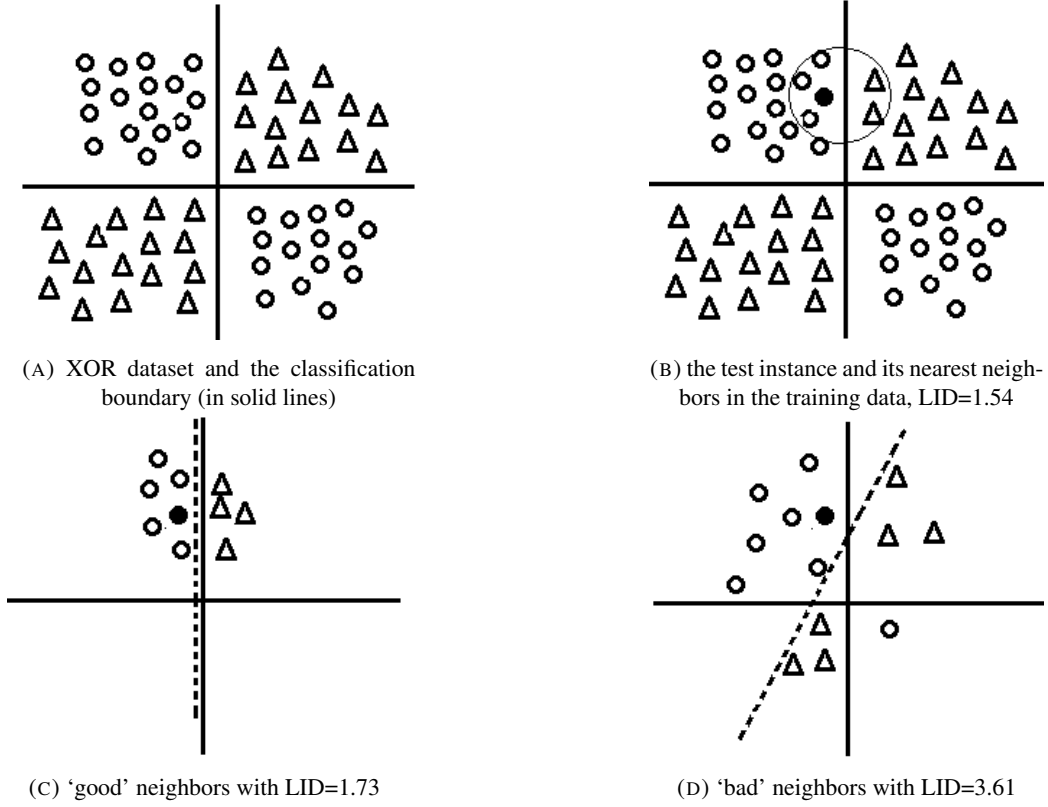


FIGURE 6.1: Illustration of a ‘good’ neighborhood and ‘bad’ neighborhood (triangles and circles are different labels). Given a test instance (filled circle) and the underlying classification boundary (solid line), generating ‘good’ neighbors (left) helps to mimic a local model (dashed line) that is similar to the true local model, while a local model (right) from ‘bad’ neighbors is far away from the true model. LID (local intrinsic dimensionality) is a possible metric for the quality of the neighbors.

away from the true classification boundary (as shown in Figure 6.1). One reason that an instance is difficult to explain is that there are too many ‘bad’ synthetic neighbors and thus explanation extraction methods fail to detect the true local classification boundary. Therefore, a synthetic neighborhood of poor quality can result in an explanation of poor quality.

Our goal in this chapter is to generate ‘good’ synthetic neighbors and avoid ‘bad’ ones to find a neighborhood of high quality, thus improving the quality of the explanation for a prediction. To facilitate this, we employ a measure of the local intrinsic dimensionality of the data, LID (Houle, 2017a), which can be regarded as capturing the number of latent variables required to represent the distribution of data within the vicinity of a reference location. As the LID rises, the number of parameters required in modeling the local distribution rises, which in turn indicates that the complexity of explaining the local model is also rising.

The LID measure has been successfully applied in similarity search and anomaly detection (Houle, 2017a; Ma et al., 2018) to characterize the complexity of the sub-manifold surrounding a test instance. In this chapter, we analyze the average LID of the synthetic neighbors of a test instance from its neighborhood in the training data, to study the difficulty of its explanation. We find that when the average LID of the synthetic neighbors is largely different from the LID of the test instance, the neighborhood is unlikely to be ‘good’, hence the instance is difficult to explain, (as shown in Figure 6.1). Based on this

observation, we propose a new approach for generating neighbor instances of high quality. The contributions of our chapter include:

- We investigate the importance of the generation of an appropriate neighborhood for local explanation methods.
- We analyze the relationship between local intrinsic dimensionality (LID) and the quality of explanations.
- We propose an effective neighbor generation method based on LID, that facilitates local explanations of high quality.

6.2 Related work

Locality of synthetic neighbors.

Generation of a synthetic neighborhood for a test instance that needs explanation is a necessary step in most local explanation methods. The realism of the synthetic neighborhood is related to the accuracy of the local model/surrogate that is fitted on the synthetic neighborhood, thus determining the quality of the local explanation extracted from the local model. LIME (Ribeiro, Singh, and Guestrin, 2016c) employs a random perturbation technique that draws the synthetic neighbors from a normal distribution centered on the test instance. A risk here is that random perturbation may hide features with local influence and benefit those with a global influence, and thus the extracted explanations may reflect global influences rather than local. In (Fong and Vedaldi, 2017), image data is perturbed through deletion of pixels; however, this technique cannot be directly extended to general tabular data, as deletion of features is problematic in this setting: tabular data models requires a fixed number of features, and setting missing feature values (the basic strategy for deleting a feature) can greatly alter the results of distance-based comparisons. In (Laugel et al., 2018), a nearest neighborhood (NN) based perturbation method is proposed that generates synthetic instances within a hypersphere whose radius is determined by the nearest instance of the opposite class label in the training data. Locality is also important in adversarial learning, as adversarial examples are designed to be close to a test instance. A common way to find an adversarial example is the gradient-based method that is introduced in fast Gradient Sign Method (FGSM) (Goodfellow, Shlens, and Szegedy, 2014) and its variations. In FGSM, an adversarial example x' of a given instance x is computed by $x' = x + \epsilon \cdot \text{sign}(\Delta g(x))$, where $\Delta g(x)$ is the gradient of the cost function of the model at x , and ϵ is the magnitude of the perturbation. FGSM differs from local explanation methods in that only one neighbor instance is generated, usually along the direction towards a local optimum.

6.3 Preliminaries

We revisit some important concepts from Section 2.3 that will be needed in this chapter, and the concept of local intrinsic dimensionality is also discussed.

Explanation. An *explanation* for the prediction of a D -dimensional instance by a given model is a D -dimensional vector where each entry represents the contribution/importance of the corresponding attribute/feature.

We allow two representations for explanations, namely **numeric explanations** and **binary explanations**.

Given a D -dimensional dataset, a numeric explanation is an explanation in the form of a weight vector as a D -dimensional vector $e = \{e_1, e_2, \dots, e_D\}$, where $e_i \in \mathbb{R}$.

In a numeric explanation, every entry indicates the contribution of the corresponding feature. The higher e_i is, the higher the contribution of the i th feature.

Given a D -dimensional dataset, a binary explanation is an explanation in the form of a bit vector as a D -dimensional vector $e = \{e_1, e_2, \dots, e_D\}$, where $e_i \in \{0, 1\}$.

In a binary explanation, the values of the entries simply indicate whether the corresponding feature is relevant or not. For example, for a prediction made by a decision tree, only the features that occur on the prediction path are relevant.

Local explanation. A *local* explanation is an instance-level explanation that explains the behavior of a classifier in the vicinity of a given instance, and it directly discriminates the given instance from its neighbors. It differs from a *global* explanation, which explains the behavior of the classifier for all instances. A globally important feature is not necessarily also locally important; for example, a decision tree may be constructed using all features, but a prediction (decision tree branch) may use only a subset of the features. All features occurring with the tree may be important in a global sense, but only those features that occur on the prediction path are locally important for the prediction of an instance.

Local intrinsic dimensionality (LID). Let a function F be positive and continuously differentiable over some open interval containing $r > 0$. The *LID* of F at r is defined as:

$$\text{LID}_F(r) := r \frac{F'(r)}{F(r)} = \lim_{\epsilon \rightarrow 0^+} \frac{F((1+\epsilon)r) - F(r)}{\epsilon F(r)}, \quad (6.1)$$

whenever the limit exists. The (asymptotic) LID of F is then:

$$\text{LID}_F^* = \lim_{r \rightarrow 0^+} \text{LID}_F(r). \quad (6.2)$$

For our scenario, LID is computed with respect to the location of a given instance q , and r represents a radial distance from q . The LID model characterizes the order of magnitude of the growth of cumulative probability measure F with respect to a neighborhood of increasing radius. As it assesses the intrinsic dimensionality of the local data submanifold around the instance q , it serves to characterize the complexity (number of parameters) required by the local data model.

Several estimators exist for calculating the local intrinsic dimensionality (Amsaleg et al., 2018; Amsaleg et al., 2019). We use the maximum likelihood estimation (MLE) in our experiments as it is known to have good convergences properties and is easy to implement. Given an instance x , and its k nearest neighbors $\text{Nei}(x)$ sorted by the distance to x in ascending order, the MLE estimator of $\text{LID}(x, \text{Nei}(x))$ is:

$$\text{LID}(x, \text{Nei}(x)) = - \left(\frac{1}{k-1} \sum_{j=1}^{k-1} \log \frac{r_j(x)}{r_k(x)} \right)^{-1} \quad (6.3)$$

where $r_j(x)$ is the distance (the Euclidean distance is used in our chapter) of the j th neighbor to x .

Note that no explicit knowledge of the underlying function F is needed by the MLE estimator - this information is implicit in the distribution of neighbor distances themselves.

Information dimension: Given a set of instances $\{x_1, \dots, x_N\}$, we may also compute the quantity $\frac{1}{N} \sum_i \text{LID}(x_i, \text{Nei}(x_i))$. This characterizes the overall intrinsic dimensionality of the manifold occupied by $\{x_1, \dots, x_n\}$. In (Romano et al., 2016), it was shown that this type of average is in fact an estimator of the well known *information dimension* over the sample domain (or manifold).

6.4 Proposed Method

6.4.1 General framework of local explanations

Given a model/predictor f and an instance x , a typical framework for extracting local explanations for $f(x)$ consists of three steps: (1) generating neighbor instances of x ; (2) labelling the neighbors using f ; (3) extracting local explanations from the generated neighbors.

Existing methods mainly differ at the third step - how to extract local explanations. In contrast, this chapter investigates the impact of the generation of the neighbors in the first step.

6.4.2 Local perturbation operators

A perturbation operator is used to generate neighbors and is a map $P : \mathbb{R}^D \mapsto \mathbb{R}^D$, given a D -dimensional instance x . It is formally defined as:

$$x' = P(x, \eta) = x + \eta \quad (6.4)$$

where η is the perturbation term or the change that is applied to the given instance. Next, we will describe some common perturbation operators that differ in how η is drawn.

A simple but useful operator is the random Gaussian perturbation that is used in LIME (Ribeiro, Singh, and Guestrin, 2016c):

$$P(x, \eta) = x + \eta, \quad \eta \sim \mathcal{N}(0, 1) \quad (6.5)$$

where $\mathcal{N}$ denotes the normal distribution from which η is drawn. If the data is not normalized to the range of $[0, 1]$, η is usually multiplied by a scale term representing the feature range. One major issue with this operator is that it is likely to generate ‘bad’ neighbors located off the sub-manifold of x .

Another operator is a gradient-based sampling technique that has been widely used in the field of adversarial learning, whereby nearby instances with the opposite class label to a given instance are generated. The strategy used in FGSM (Goodfellow, Shlens, and Szegedy, 2014) is:

$$P(x, \eta) = x + \eta, \quad \eta = \epsilon \cdot \text{sign}(\Delta g(x)) \quad (6.6)$$

where $\Delta g(x)$ is the gradient of the cost function at the test instance x and ϵ is the magnitude of perturbation.

Recently, a nearest neighbor (NN) based method was proposed in (Laugel et al., 2018). It first finds the nearest instance x_c of the given test instance x in the training data such that $f(x) \neq f(x_c)$, where f is the prediction model. The perturbation is then

$$\begin{aligned} P(x, \eta) &= x + \eta, \quad \eta \sim \text{Unif}(\text{range}(x)) \\ \text{s.t. } \text{dist}(x, P(x, \eta)) &\leq \text{dist}(x, x_c) \end{aligned} \quad (6.7)$$

where $\text{dist}(x, z)$ measures the distance of two instances x and z , $\text{range}(x)$ denotes the possible value ranges of the features, and Unif denotes the uniform distribution.

6.4.3 Learning local explanations

Given a black-box model f and an instance x , the target of learning the local explanation is to find a local model g from a hypothesis set $\mathcal{G}$ in the neighborhood of x . The process

can be formulated as:

$$\begin{aligned} & \underset{g \in \mathcal{G}, x_i = P(x, \eta_i)}{\text{minimize}} && \frac{1}{m} \sum_{i=1}^m \mathcal{L}(g(x_i), f(x_i)) + \alpha \mathcal{R}(g) \\ & \text{subject to} && \text{Locality_Constraint}(x_i), i = 1, \dots, m. \end{aligned} \quad (6.8)$$

where $\mathcal{L}$ is the loss function, $\mathcal{R}(g)$ is an interpretability regularization term to ensure that the chosen g is interpretable, α is the regularization rate, $P(x, \eta)$ is the local perturbation operator that generates the synthetic neighborhood of x , and $\text{Locality_Constraint}(x_i)$ is a constraint on the generated neighborhood to ensure locality fidelity of the local explanations.

The main challenges for solving this problem are: (1) interpretability constraints are subjective and difficult to formulate; (2) the underlying local classification boundary is unknown and thus it is difficult to formally define the locality constraints.

In existing methods (Ribeiro, Singh, and Guestrin, 2016; Ribeiro, Singh, and Guestrin, 2018; Fong and Vedaldi, 2017), the first challenge has been addressed by restricting $\mathcal{G}$ to a set of interpretable models $\mathcal{G}^*$ (such as decision trees, rules or linear models). We therefore focus on the second issue, and approach it from the perspective of local intrinsic dimensionality, by defining an LID based locality constraint.

LID can be used to characterize the complexity of the neighborhood (a set of instances) surrounding a test instance (single instance). Given a model f and a test instance x , assume there exists a true neighborhood $Nei^*(x)$ of x , such that we can train a local model g that is a perfect local surrogate for f , without incurring an increase in local intrinsic dimensionality. When using local perturbation operators to generate a synthetic neighborhood $Nei_s(x)$, in order to extract a faithful explanation from $Nei_s(x)$, we want the LID of $Nei_s(x)$ to be as close as possible to that of $Nei^*(x)$. Here, we use the average LID of the neighborhood (an estimator of information dimension (Romano et al., 2016), or correlation dimension (Houle, 2017b)) as an estimator for the divergence (dissimilarity) of the explainability of $Nei_s(x)$ and $Nei^*(x)$:

$$\begin{aligned} \text{divg}(Nei_s(x), Nei^*(x)) &= \frac{1}{|Nei_s(x)|} \sum_{x_i \in Nei_s(x)} \text{LID}(x_i, Nei^*(x_i)) \\ &\quad - \frac{1}{|Nei^*(x)|} \sum_{x_j \in Nei^*(x)} \text{LID}(x_j, Nei^*(x_j)) \end{aligned} \quad (6.9)$$

As $Nei^*(x)$ is the true neighborhood of x , we assume that any instance in $Nei^*(x)$ should have a similar LID to x (this natural assumption has been recently used in the development of tight estimators of LID (Amsaleg et al., 2019)). Under this assumption, we can approximate $\frac{1}{|Nei^*(x)|} \sum_{x_j \in Nei^*(x)} \text{LID}(x_j, Nei^*(x_j))$ by $\text{LID}(x, Nei^*(x))$. The divergence becomes:

$$\begin{aligned} \text{divg}(Nei_s(x), Nei^*(x)) &\approx \frac{1}{|Nei_s(x)|} \sum_{x_i \in Nei_s(x)} \text{LID}(x_i, Nei^*(x_i)) \\ &\quad - \text{LID}(x, Nei^*(x)) \end{aligned} \quad (6.10)$$

As $Nei^*(x)$ is unknown, we approximate it using k nearest neighbors of x in the training data, which is denoted as $Nei_k(x)$. Then the empirical divergence is:

$$\begin{aligned} \overline{\text{divg}}(Nei_s(x), Nei^*(x)) &\approx \frac{1}{|Nei_s(x)|} \sum_{x_i \in Nei_s(x)} \text{LID}(x_i, Nei_k(x_i)) \\ &\quad - \text{LID}(x, Nei_k(x)) \end{aligned} \quad (6.11)$$

In the ideal case $\text{divg}(\text{Nei}_s(x), \text{Net}^*(x))$ is close to 0 and we have:

$$\frac{1}{|\text{Nei}_s(x)|} \sum_{x_i \in \text{Nei}_s(x)} \text{LID}(x_i, \text{Nei}_k(x_i)) \approx \text{LID}(x, \text{Nei}_k(x)) \quad (6.12)$$

Now, we can formally refine the optimization problem as:

$$\begin{aligned} & \underset{g, x_i = P(x|\eta_{n,f})}{\text{minimize}} && \frac{1}{m} \sum_{i=1}^m \mathcal{L}(g(x_i), f(x_i)) \\ & \text{subject to} && \left| \frac{1}{m} \sum_{i=1}^m \text{LID}(x_i, \text{Nei}_k(x_i)) - \text{LID}(x, \text{Nei}_k(x)) \right| \leq \epsilon \end{aligned} \quad (6.13)$$

where g is chosen from the interpretable model set $\mathcal{G}^*$, the term $\text{LID}(x_i, \text{Nei}_k(x_i))$ measures the LID of a generated instance x_i with respect to its k nearest neighbors in the training data, and ϵ is a sufficiently small value. Intuitively, the LID constraint checks whether the LID of the test instance x is equal to the information dimension of its generated neighborhood. For a ‘good’ neighborhood, we expect these terms to be equal.

There isn’t, however, an available closed form solution for the above optimization problem. Thus we propose a heuristic method to find an approximate solution by learning a local subspace of appropriate local intrinsic dimensionality around x , and applying random perturbations within this local subspace.

6.4.4 LEAP - Local embedding aided perturbation

Random perturbations can introduce noise that push an instance far from its underlying local sub-manifold, thereby effectively increasing the number of parameters required for an adequate local model. The LID distribution of instances in the synthetic neighborhood may then be very different to the LID of the test instance. Our proposed method mitigates the effect of noisy perturbations that do not conform to the local manifold, by first approximating a local embedding/subspace around the test instance and then generating the neighbors within the local embedding. So as not to increase the complexity of the local data model, the LID of the test instance is used as the target dimensionality for the embedding. Since we uniformly generate synthetic neighbors within the local embedding, the newly generated instances can be expected to preserve the LID of the test instance. Our proposed method (LEAP - Local embedding aided perturbation) is shown in Algorithm 8. The details are described as follows:

Inputs: x is the test instance whose prediction needs explanation. M is the required number of instances to be generated in the vicinity of x . T is the training data on which the given model is trained. k is the number of nearest neighbors of x used for LID estimation. λ is the dimension relaxation for the approximation of LID.

Output: Z is the set of M generated instances.

Process:

Step 1 (Lines 1-3). The initial step is to fit a local embedding in the vicinity of x . We first select a set of neighbor instances of x from the training data using k-nearest-neighbor search. Then a local embedding is extracted from these instances using a subspace learning technique (we use PCA in the experiments, but kernel PCA or LDA are alternative possibilities) using $\text{LID}(x, \text{Nei}_k(x))$ as the target dimensionality for the embedding, and a dimension relaxation λ (small λ may miss some important features thus result in low recall of explanations, and large λ can introduce noisy features thus result in low precision of explanation, an appropriate value of 5 is suggested by the author of (Houle, 2017a)) is added to the target dimensionality as $\text{LID}(x, \text{Nei}_k(x))$ is just an approximation.

Step 2 (Lines 4-5). After the local embedding has been found, the given instance x is then projected to $\hat{x}$ on the local embedding.

Step 3 (Lines 6-11). The last step is to generate the required number of instances to form the local neighborhood of x . At each generation step, the random perturbation operator is applied to $\hat{x}$ to get $\hat{z}$ and then the newly generated instance is mapped back into the original feature space, and its weight is (optionally) set according to the distance between $\hat{z}$ and $\hat{x}$. These weights can be optionally used by explanation extraction methods.

Algorithm 8: LocalEmbeddingPerturbation

Input: Instance x , number of generated neighbors M , training data T , KNN threshold k , dimension relaxation λ

Output: $Z = \{z_1, z_2, \dots, z_M\}$

```

1  $nei = \text{getkNearestNeighbors}(x, T, k);$ 
2  $lid = \text{LocalIntrinsicDim}(x, nei);$ 
3  $le = \text{new LocalEmbedding}(nei, lid) + \lambda;$ 
4  $\hat{x} = le.\text{project}(x);$ 
5  $Z = \{\};$ 
6 while  $|Z| < M$  do
7   uniformly generate an instance  $\hat{z}$  within the local embedding;
8    $z = le.\text{projectToOriginal}(\hat{z});$ 
9    $z.\text{setWeight}(\exp(-\text{dist}(\hat{z}, \hat{x})));$ 
10   $Z = Z \cup \{z\};$ 
11 end
12 return  $Z;$ 
```

6.4.5 Local explanation extraction with LEAP

For any local explanation extraction method (e.g., LIME) that needs to generate local neighbors, LEAP can be directly used to replace the generation process. We take the given model f and the parameters that are required by LEAP as inputs, and return an explanation for the prediction of the given x by f . First, the neighbor instances of x are generated using LEAP. Then, all new instances are labeled by the given model f . Finally, the local explanation is generated using a technique like LIME (e.g. with a linear model or a decision tree). The process is shown in Algorithm 9.

Algorithm 9: Explanation_LEAP

Input: Model f , instance x , number of generated neighbors M , KNN threshold k , training data D_{train}

Output: e

```

1  $Z = \text{LocalEmbeddingPerturbation}(x, M, k, D_{train}, C);$ 
2 for  $z \in Z$  do
3    $z.\text{label} = f(z);$ 
4 end
5  $e = \text{extractExplanation}(x, Z);$ 
6 return  $e;$ 
```

6.5 Experiments

In this section, we define the metrics that will measure the quality of the explanations, and experimentally evaluate the performance of our proposed method. The experimental settings are described, and then we demonstrate how the true explanations, which are used as ground truth to evaluate the generated explanations, are found. Comparisons of the performance of the proposed method and that of the baselines on several datasets are reported. Finally, a case study is explained.

6.5.1 Explanation metrics

Recall from Section 2.4.1, given a model f and a D -dimensional instance x , assume $e' = \{e'_1, \dots, e'_D\}$ is the true model explanation, and $e = \{e_1, \dots, e_D\}$ is the explanation generated by an explanation system (e.g., LIME). The quality of e is measured by its closeness to e' . Given two representations of explanations, we define the following metrics.

When e, e' are numeric explanations, where $e_i, e'_i \in \mathbb{R}$, the quality of e , denoted by $Q(e|e')$, is measured using the cosine similarity:

$$Q(e|e') = |\text{cosine_similarity}(e|e')| = \left| \frac{e \cdot e'}{|e| |e'|} \right| \quad (6.14)$$

where $e \cdot e'$ is the dot product of two explanations, and $|e|$ calculates the l^2 -norm of e .

The closer $Q(e|e')$ is to 1, the higher the quality of e , and if it is 1 then e and e' are parallel.

When e, e' are binary explanations, where $e_i, e'_i \in \{0, 1\}$, $Q(e|e')$ is measured by the F_1 score:

$$Q(e|e') = \frac{2 * \text{recall}(e|e') * \text{precision}(e|e')}{\text{recall}(e|e') + \text{precision}(e|e')} \quad (6.15)$$

where $\text{recall}(e|e') = \frac{\sum_{i=1}^D e_i * e'_i}{\sum_{i=1}^D e'_i}$, which indicates how many chosen features (with non-zero explanation values in e) are truly important (with non-zero explanation values in e'), and $\text{precision}(e|e') = \frac{\sum_{i=1}^D e_i * e'_i}{\sum_{i=1}^D e_i}$, which indicates how many truly important features are chosen in e . $Q(e|e')$ equals 1 if and only if e and e' are identical, and is less than one otherwise.

6.5.2 Settings

We conduct our experiments on six synthetic datasets and nine UCI datasets. The proposed neighborhood generation method (LEAP) is plugged into LIME, and compared with three baselines for the numeric explanation case and binary explanation case. The local neighbor generating methods we have compared are described as follows:

LEAP(N) - Numeric explanation method with the proposed local embedding perturbations and local linear models.

Random(N) - Numeric explanation method with random perturbations (Eq. (6.5)) and linear models, which is the default method used in LIME.

NN(N) - Numeric explanation method with nearest neighbor perturbations (Eq. (6.7)) and local linear models.

Gradient(N) - Numeric explanation method with gradient-based perturbations (Eq. (6.6)) and local linear models.

LEAP(B) - Binary explanation method with the proposed local embedding perturbations and decision trees.

Random(B) - Binary explanation method with random perturbations (Eq. (6.5)) and decision trees.

NN(B) - Binary explanation method with nearest neighbor perturbations (Eq. (6.7)) and decision trees.

Gradient(B) - Binary explanation method with gradient-based perturbations (Eq. (6.6)) and decision trees.

Parameter optimization via a nested validation process during the training process is a common technique for classification tasks; however, it presents difficulties for explanation systems, due to the fact that true model explanations are always unavailable when deploying such systems in real-life applications. One possible solution is to generate a parameter setting using validation datasets whose true model explanations are available, and then to apply the same setting for any new dataset. To simulate such a process, we apply LEAP with different parameter settings of two major parameters K and M on four separate UCI datasets (*crx*, *ILPD*, *sick* and *sonar*, which are different from those used for evaluation) as validation datasets, then the recommended parameter setting ($K = 5D$ where D is the dimensionality of data, and $M=500$) (that achieves the best faithfulness score on average over the validation datasets) is used for all the datasets whose results are reported.

6.5.3 Synthetic data and true explanations

The synthetic datasets are generated using synthetic functions described below as the classifiers, and true explanations are then directly generated from these functions:

- $f_1(x) = \begin{cases} x[1] - 4x[2] + 2x[3] + 3, & x[1] \leq 10 \\ -2x[1] - 3x[2] + x[2] - 2, & 10 < x[1] \leq 20 \\ 3x[1] + x[2] - 2x[3] + 2, & x[1] > 20 \end{cases}$
where $x[i] \in [0, 30]$.
- $f_2(x) = (x[1])^3 - 2(x[2])^2 + 3x[3]$, where $x[i] \in [-100, 100]$.
- $f_3(x) = x[1] - x[2] \sin(x[2])^2$ where $x[i] \in [-10, 10]$.
- $f_4(x) = ((x[1] == 1) \wedge x[2] \wedge x[3]) \vee ((x[1] == 2) \wedge x[4] \wedge x[5]) \vee ((x[1] == 3) \wedge x[6] \wedge x[7]) \vee ((x[1] == 4) \wedge x[8] \wedge x[9])$, where $x[1] \in [1, 4]$ and $x[2], \dots, x[9]$ are binary attributes.

For each synthetic dataset, the attribute values are drawn from a uniform distribution, and for $i = 1, 2, 3$, the class label is 1 if $f_i(x) > 0$ else 0 otherwise. For $i = 4$, the class label is 1 if $f_4(x) = \text{true}$ else 0 otherwise. Six synthetic datasets (two for numeric explanation scenarios and the other two for binary explanation scenarios) are generated as follows: for the numeric functions f_i ($i = 1, 2, 3$), the true explanation of a given x is the gradient of the curve $f_i = 0$ at the closest point on the curve from x . This is because we expect the true explanation to be like a locally linear model. For the binary explanation function f_4 , the true explanation is defined below.

- **Synthetic-1** consists of three attributes ($x[1], x[2], x[3]$) and a class label. The function f_1 is used to label the instances. Given an instance x , the true local (numeric)

explanation is defined as:

$$e_{true}(x) = \begin{cases} (1, -4, 2), & x[1] \leq 10 \\ (-2, -3, 1), & 10 < x[1] \leq 20 \\ (3, 1, -2), & x[1] > 20 \end{cases}$$

- **Synthetic-2** consists of ten attributes $(x[1], x[2], \dots, x^{(10)})$ and a class label. The function f_1 used in Synthetic-1 is also used to label the instances such that the other seven attributes are noise and not used in the prediction process. The true explanations of the instances are defined similarly to Synthetic-1 dataset.
- **Synthetic-3** consists of three attributes $(x[1], x[2], x[3])$ and a class label. The function f_2 is used to label the instances. Given an instance x , the true local (numeric) explanation is defined as:

$$e_{true}(x) = (3(x^*[1])^2, -4x^*[2], 3)$$

, where x^* is the closest point of x to the function $f_2 = 0$.

- **Synthetic-4** consists of three attributes $(x[1], x[2])$ and a class label. The function f_3 is used to label the instances. Given an instance x , the true local (numeric) explanation is defined as:

$$e_{true}(x) = (1, -\sin(x[2])^2 - 2(x[2])^2 \cos(x[2])^2)$$

, where x^* is the closest point of x to the function $f_3 = 0$.

- **Synthetic-5** consists of nine attributes $(x[1], x[2], \dots, x[9])$ and a class label. The function f_4 is used to label the instances. Given an instance x , the true local (binary) explanation is defined as:

$$e_{true}(x) = \begin{cases} (1, 1, 1, 0, 0, 0, 0, 0, 0), & x[1] = 1 \\ (1, 0, 0, 1, 1, 0, 0, 0, 0), & x[1] = 2 \\ (1, 0, 0, 0, 0, 1, 1, 0, 0), & x[1] = 3 \\ (1, 0, 0, 0, 0, 0, 0, 1, 1), & x[1] = 4 \end{cases}$$

- **Synthetic-6** consists of twenty attributes $(x[1], \dots, x[20])$ and a class label. The function f_4 used in Synthetic-3 is also used to label the instances such that other eleven attributes are noise and not used in the prediction process. The true explanations of the instances are defined the same way as in the Synthetic-5 dataset.

6.5.4 UCI data and true explanations

We also conduct experiments on nine UCI datasets. For the purpose of evaluating the numeric explanations, all nominal attributes are converted to continuous and all attributes are normalized into the range $[0,1]$. Two interpretable classifiers (from which we can get the true model local explanations) are used as the black-box for local explanation extraction methods. We use logistic regression and decision trees as used in (Ribeiro, Singh, and Guestrin, 2016c), and the true explanations are extracted as follows

- **Logistic regression.** The true explanations are numeric explanations and are obtained directly from the model parameter (the coefficients). For a D -dimensional

instance x , $e_{true}(x) = (|\omega_1|, |\omega_2|, \dots, |\omega_D|)$, which is the vector of the coefficients and ω_i is the coefficient for the i th attribute.

- **Decision tree.** The true explanations are binary explanations and are obtained from the structure of the tree. For a D -dimensional instance x , $e_{true}(x) = (e_1, e_2, \dots, e_D)$, e_i is 1 if the i th attribute occurs in the prediction path from the root to the leaf, and 0 otherwise.

The true explanations are extracted in the same way as in LIME: decision trees are trained from the training data, and the true explanations for the test data are obtained from the trees, though the explanation systems treat the decision trees as black boxes. Decision trees are trained for all datasets. The true explanation for the prediction of an instance is the set of features that occur along the path from the root to the leaf corresponding to the prediction.

6.5.5 Evaluation on synthetic data

The baselines are first compared on the synthetic datasets. The results (Table 6.1) show that the local explanation method with LEAP achieves the highest performance across these datasets. In particular, LEAP is more robust to non-linear classification boundaries as the explanation quality of LEAP is much higher than baselines on synthetic-3 and synthetic-4 datasets. The reason will be discussed in details in Section 6.5.7.

6.5.6 Evaluation on UCI data

We train two white-box classifiers - logistic regression and decision trees - on the UCI datasets. Then the local explanation method (LIME framework) with different neighborhood generation methods is used to extract explanations for the instances in the test datasets, while treating these interpretable classifiers as black-box classifiers. Specifically, LIME based on local linear models is used to evaluate the numeric explanations for logistic regression, and LIME based on local decision trees is used to evaluate binary explanations. The results in (Table 6.2) show that LEAP has excellent performance in comparison to the three baselines. In particular,

6.5.7 Visualization of the quality of explanations for different methods

We conduct an experiment on a 2D synthetic dataset (we use 2D dataset for visualization purposes), generated using the function $f(x[1], x[2]) = \text{sign}(x[1] - x[2] \sin(x[2])^2)$, where the true explanation for a point is the gradient (represented as a weight vector) of the closest point on the classification boundary ($x[1] = x[2] \sin(x[2])^2$) to the test point being explained. The quality of explanations for different neighborhood generating methods is shown in Figure 6.2. For Random(N) (used in LIME), an instance that is close to the classification boundary is likely to be difficult to explain. This observation also suggests that if an instance is far away from the classification boundary, finding an explanation may be more obvious. LEAP and NN are particularly effective for instances close to the classification boundary, with LEAP having an edge over NN in average explanation quality. For the gradient-based method, the quality of the explanations is similar to Random, and worst when the instance is far away from the boundary.

In terms of the LIDs, while the mean of the LID of the test instance in its vicinity is 1.37, the average LIDs of the synthetic neighborhoods are: 1.4 for LEAP, 18.4 for Random, 2.4 for NN and 6.9 for Gradient. It shows that LEAP achieves the closest LID as the test instance.

Dataset	Random(N)	Gradient(N)	NN(N)	LEAP(N)	Random(B)	Gradient(B)	NN(B)	LEAP(B)
Synthetic-1	0.88	0.65	0.82	0.97	-	-	-	
Synthetic-2	0.82	0.71	0.78	0.90	-	-	-	
Synthetic-3	0.71	0.72	0.85	0.94	-	-	-	
Synthetic-4	0.65	0.61	0.75	0.90	-	-	-	
Synthetic-5	-	-	-	-	0.87	0.78	0.76	0.92
Synthetic-6	-	-	-	-	0.81	0.62	0.74	0.88

TABLE 6.1: Explanation quality on synthetic datasets (the best value in bold; ‘-’ means method does not apply on corresponding dataset, e.g., Synthetic-1 is for numeric explanation and binary explanation method does not apply).

Dataset	Random(N)	Gradient(N)	NN(N)	LEAP(N)	Random(B)	Gradient(B)	NN(B)	LEAP(B)
	logistic regression (numeric explanations)				decision trees (binary explanations)			
Adult	0.84	0.63	0.72	0.90	0.64	0.61	0.57	0.73
Balloon	0.93	0.82	0.54	1.00	0.92	0.81	0.50	0.94
Blood	0.89	0.85	0.61	0.96	0.94	0.58	0.92	0.97
Breast-cancer	0.82	0.78	0.59	0.87	0.65	0.57	0.50	0.70
Diabetes	0.87	0.74	0.62	0.99	0.80	0.47	0.71	0.80
Hepatitis	0.72	0.68	0.56	0.80	0.82	0.38	0.72	0.87
Iris	0.91	0.82	0.64	0.94	1.00	1.00	1.00	1.00
Ionosphere	0.91	0.82	0.74	0.92	0.92	0.66	0.89	1.00
Labor	0.86	0.75	0.64	0.89	0.88	0.64	0.77	0.95
Titanic	0.85	0.83	0.59	0.87	0.74	0.73	0.66	0.79
Vote	0.90	0.77	0.62	0.93	0.82	0.66	0.50	0.89

TABLE 6.2: Explanation quality on UCI datasets (the best value is highlighted in bold).

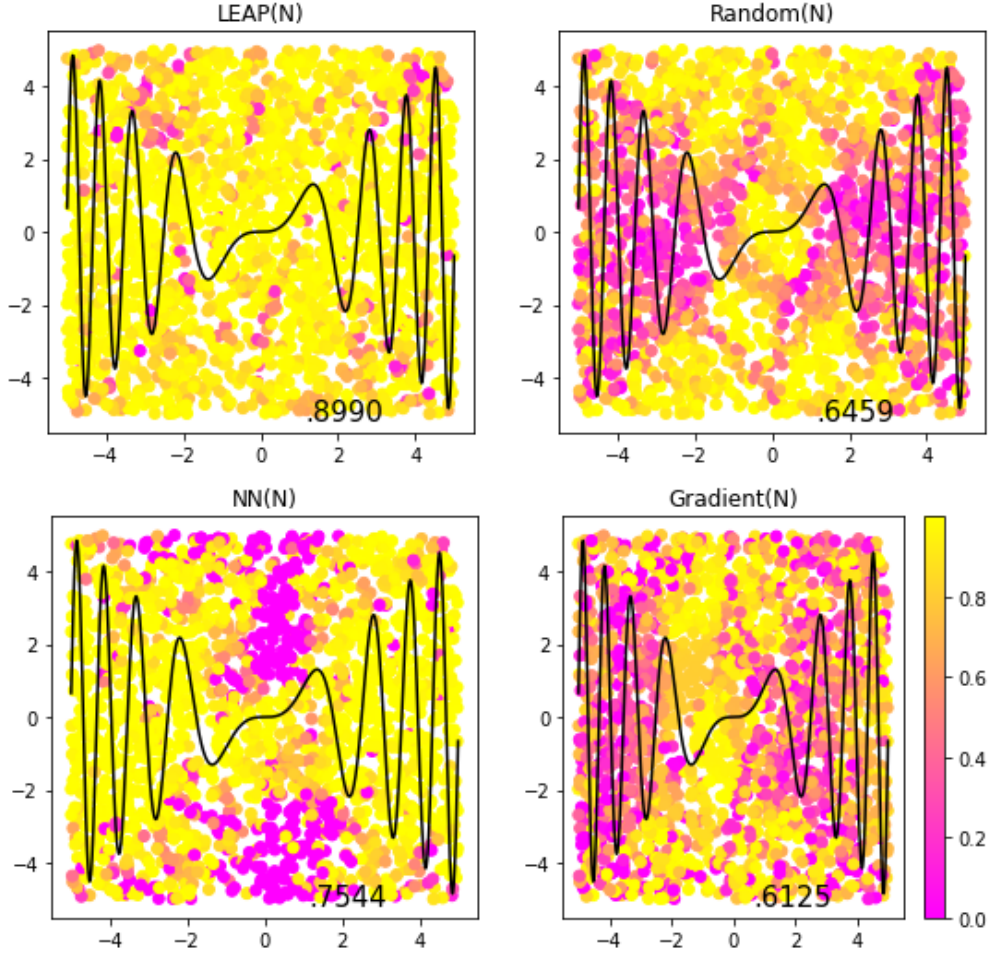


FIGURE 6.2: Demonstration of difficulty of explanation for different neighborhood generating methods. The black solid line is the classification boundary, and difficulty (explanation quality) varies from difficult (red) to easy (yellow). The average explanation quality is shown in the bottom-right of each figure.

We also demonstrate some synthetic neighborhoods that are generated by different methods in Figure 6.3.

6.5.8 Locality analysis

We compare the locality for different baselines. The same data and ground truth explanations as in Section 6.5.7 are used, and we adopted the locality measurement that was proposed in (Laugel et al., 2018). Given a test instance and the local model that is used to extract the local explanations, a set of instances is drawn from a normal distribution centered on the test instance with variance σ . σ is treated as a radius, and the average accuracy for a local model applied to the set of test instances is defined as the locality at radius σ . Results for the locality of the baselines are shown in Figure 6.4. It can be seen that LEAP achieves the highest locality for smaller radius, which is expected. On the other hand, the locality of Random(N) is nearly uniform for all radii, illustrating that LIME+Random(N) is impacted by global fidelity (a finding which accords with that of (Laugel et al., 2018)). It is desirable for the locality to drop with increasing radius, because our target is to achieve a ‘local’ explanation model. For NN and Gradient, a similar

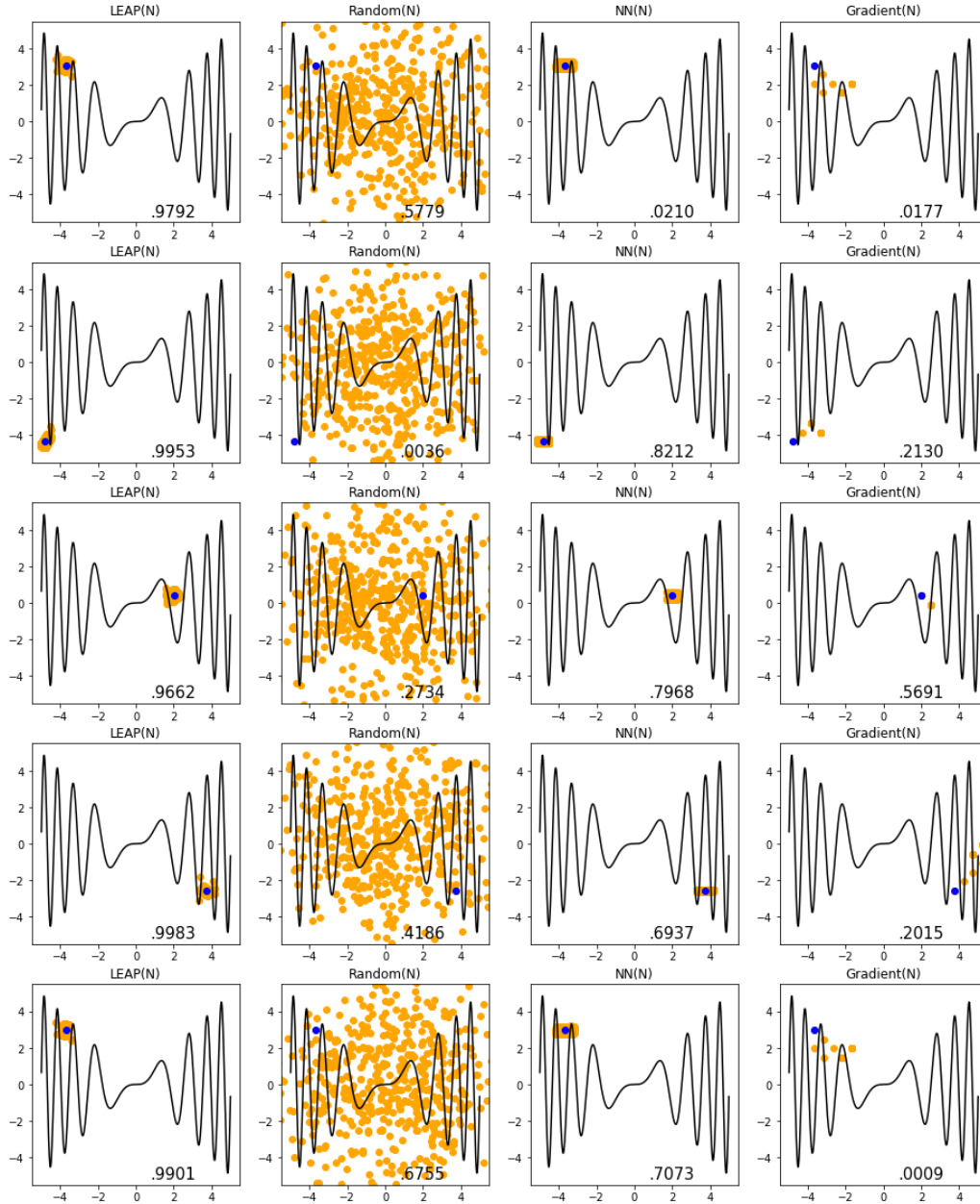


FIGURE 6.3: Demonstration of the synthetic neighborhoods that are generated by different methods. The test instances are shown in blue color, and the synthetic neighborhoods are shown in orange color. The quality of explanations extracted from the synthetic neighbors is shown in the bottom-right.

decreasing trend is evident, but the locality (accuracy of the local models) is lower than LEAP.

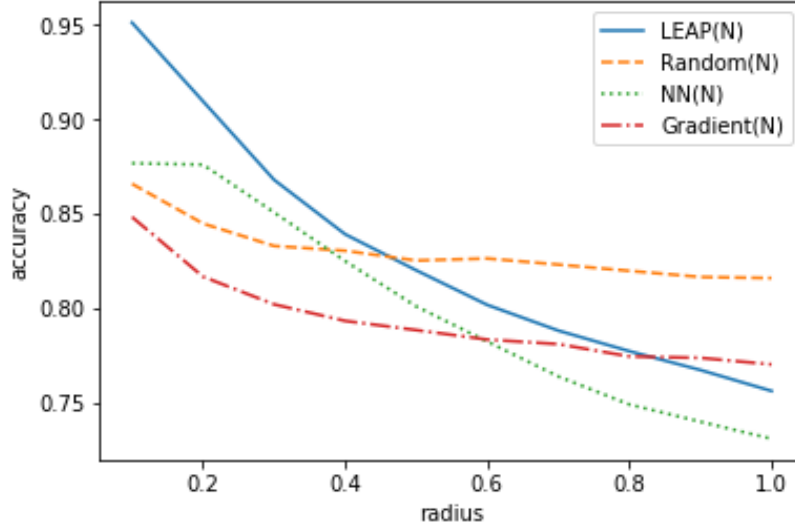


FIGURE 6.4: Locality analysis (plot of the accuracy of the local models when increasing the radius)

6.5.9 Example: Why does the quality of explanations matter?

We have proposed LEAP to improve the quality of the local explanations, and we next provide an example of how an explanation having higher quality can be beneficial for taking an action. Our study is performed in the context of spam classification, on a dataset containing 1956 comments from 5 different YouTube videos (Uysal, 2018). We split it to two groups with 70% for training and 30% for testing. The class label is whether or not the comment is a spam.

We train a Random Forest (with 100 trees) and extract explanations for test comments using different synthetic neighborhood generation methods. Each explanation e for a test instance x is evaluated in the following way: based on the explanation e , x is modified so that the occurrence of the word that most supports the opposite class label is increased 10 times. If the class label of the modified instance is flipped (when the modified instance is passed to the random forest), we say the explanation e is useful, since it facilitated a perturbation to be made on the instance that had an intuitive effect. Conversely, if the class label is not flipped, then the explanation is not useful, since the perturbation didn't operate as expected.

For this task, Random(N) produces 87.34% useful explanations while LEAP produces 95.40% useful explanations. An example of where Random(N) fails to find a useful explanation but LEAP succeeds, is shown in Table 6.3. Random(N) finds the word 'the' is the most supportive feature for class label 'Not spam' but the corresponding modification is still classified as 'Spam'. LEAP finds the word 'projects' and the corresponding modification fools the classifier successfully. Although humans may regard both modifications as spam, the explanation of LEAP reveals how the classifier 'thinks'.

Method	Explanations	Modified Comment using explanation	Classification
Random(N):	Spam: {http, and, ref, you}; Not spam: {The, here, Effects}	The The The The The The The The The The The projects After Effects, Music, Foto, Web sites and another you can find and buy here http...	Spam (label of Modified comment not flipped as expected)
LEAP:	Spam: {http, ref, Music, buy}; Not spam: {projects, Effects, find}	projects projects projects projects projects projects projects The projects After Effects, Music, Foto, Web sites and another you can find and buy here http...	Not spam (label of Modified comment is flipped as expected)

TABLE 6.3: Explanations extracted with Random(N) and LEAP for the comment *"The projects After Effects, Music, Foto, Web sites and another you can find and buy here http..."* which is classified as Spam

6.6 Conclusion

In this chapter, we proposed a new approach for neighborhood generation, which is a critical phase in local explanation systems. Our approach is based on consideration of the local intrinsic dimensionality of the instance whose predicted label is being explained. Our method can be used as a modular procedure in any existing local explanation method that requires neighborhood generation. We compared our proposed technique with the baselines that are used in local explanation methods, and demonstrated it achieves excellent performance when compared with state of the art alternatives.

Chapter 7

Learning with Expert Supervision to Make Reliable Predictions

7.1 Introduction

Recent advances to improve the interpretability of data mining models have taken two major directions: (1) building new interpretable models that can achieve high accuracy, and (2) improving the interpretability of complex models, i.e., by providing explanations for their predictions. In both cases, interpretability can be viewed as a supplementary feature, which does not directly influence model learning, since the training process solely focuses on optimising accuracy. Consequently, there is no guarantee that such models will make predictions which accord with domain knowledge. In this chapter, we propose the notion of reliability, which refers to the ability of a model to make a prediction (reach a decision) for an instance based on features that match expert domain knowledge about that instance. We develop explanation-based partition-wise linear models (xPWL) that can make reliable as well as accurate predictions. xPWL uses extra information - expert explanations for training instances - as expert supervision within the learning process to help improve the reliability of the model. Contrast patterns are mined to partition the dataset and used to represent local partitions. Every pattern is associated with a local linear model. The overall prediction for a given instance is performed by generating an instance-specific model from the local models associated with the matching patterns. Experimental results demonstrate that xPWL is able to achieve competitive accuracy compared with the state-of-the-art baselines, while achieving superior reliability. A further property we propose in this section is the notion of prediction reliability. We say a prediction on an instance is reliable if its explanation matches a gold standard (expert) explanation. In particular, a prediction about an instance is reliable if the features occurring in its explanation match with features that a domain expert considers to be the true explanation for the instance.

Explanation reliability measures how close the logical reasoning behind a model is to the experts' domain knowledge and reflects how reliable a prediction is. We believe that the explanation reliability of a prediction can be as important as accuracy in situations where the lack of reliability results in rejection of the prediction or has severe consequences. For example, suppose a streaming video user likes comedy movies but does not care about the actors (assume that the system has such information by preference setting), and the video website gives the user a movie recommendation with an explanation that it stars the same actor who appeared in a movie the user just viewed. Meanwhile, another video website gives the same movie recommendation with an explanation that it is the same type of comedy as the user's last viewed movie. For the same prediction but with different explanations, the user is more likely to take the second one as the explanation is more compelling. Another example is a medical scenario, where doctors may administer treatments or actions based on the explanations of a prediction made by a model. Though the disease diagnosis might be correct, if it is inferred from incorrect features (symptoms)

and the explanation consists of these features, treatment based on these features could be adverse to the patient and may even worsen the disease.

Reliability can be a new metric for evaluating models, whereas traditional classifiers are trained to focus on accuracy, and provide no guarantee about reliability. Our research question is thus: *how can one develop a model that makes both accurate and reliable predictions?* We address this question by proposing a new classifier called explanation-

	<i>Traditional classifiers</i>		<i>Proposed Method (xPWL)</i>
<i>Inputs</i>	Training data		Training data & expert explanations
<i>Explanations</i>	Inferred from model for interpretable models	Explained by an explanation method for non-interpretable models	Inferred from the model
<i>Objective function</i>	Loss function in terms of accuracy		Loss function in terms of both accuracy and reliability

TABLE 7.1: Comparison of xPWL with traditional classifiers

based partition-wise linear models (xPWL), which incorporates expert explanations about training instances as additional expert supervision input. For example, user reviews of movies can be extracted as expert explanations for a recommendation system; the diagnosis reports from doctors can be used as expert supervision for a medical diagnosis model. Using this extra information, xPWL is able to make reliable explanations without sacrificing accuracy. A comparison of our proposed xPWL with traditional classification is shown in Table 7.1.

In summary, our contributions are:

- we propose the notion of reliability as an important property for predictions;
- we develop an algorithm, xPWL, that optimises both accuracy and reliability for its predictions, modelled as a convex optimization loss function, so that xPWL is able to provide explanations for its predictions;
- we demonstrate that xPWL can achieve greater reliability and competitive accuracy compared to several baselines.

7.2 Related work

Partition-wise models. Partition-wise models refer to models that divide the input space into multiple partitions (sometimes also called regions), where the instances in each partition behave similarly and can be predicted by a local simple model (i.e., a linear model). Partition-wise linear models (PWL) Oiwa and Fujimaki, 2014 divide the input space by parallel splitting each coordinate/feature, and assign a linear model to each partition. A prediction for a given test instance is made by a predictor that is a linear combination of active partition-wise weight vectors. Our method xPWL improves PWL in two ways: 1) instead of obtaining partitions by parallel splitting features, xPWL mines contrast patterns

as partitions, which are shown to be more effective; 2) we add an explanation based regularization term such that the model is trained to make reliable predictions. Other common partition-wise models include LSL-SP (Wang and Saligrama, 2012), LLSVM (Ladicky and Torr, 2011) and LDKL (Jose et al., 2013), and (Oiwa and Fujimaki, 2014) compares them.

Pattern aided models. Pattern aided methods have shown promising results in terms of accuracy as white-box classifiers, and they also belong to partition-wise models, as every pattern represents a subspace of the input space. Decision trees and random forests are well known examples where each path from the root to a leaf in a tree can be viewed as a pattern, and each pattern is associated with a simple predictor that always returns the label of the leaf node. Recently, a pattern aided classifier (PXC) has been proposed (Dong and Taslimitehrani, 2016). PXC first generates contrast patterns according to the “difficulty” of the dataset and then trains local classifiers on each partition associated with a pattern. CBA (Ma and Liu, 1998) and similar techniques like CAEP (Dong et al., 1999) and iCAEP (Zhang, Dong, and Ramamohanarao, 2001) extract contrast patterns from the training data, and use the most probable pattern to predict a given instance. PatClass (Cheng et al., 2007) and similar methods like DDPMine (Cheng et al., 2008), DPClass (Shang et al., 2016), HARMONY (Wang and Karypis, 2005) and those proposed in (Caruana et al., 2015)(Letham et al., 2015) extract contrast patterns as the classification rules and predict a given instance by combining the results of matching patterns. Our method xPWL differs from these methods by: 1) tuning the model parameters globally, and 2) training with explanation.

Explanation systems. Recent works have focused on one important aspect of interpretability - providing explanations for the predictions of a model. The methods proposed in (Adler et al., 2018)(Henelius et al., 2014) (Robnik-Šikonja and Kononenko, 2008) calculate a vector of weights representing the predictive powers by perturbing the feature values. LIME (Ribeiro, Singh, and Guestrin, 2016c) and its variant aLIME (Ribeiro, Singh, and Guestrin, 2018) train a local linear interpretability model in the vicinity of an instance and generate explanations from it. They are called black-box explanation systems as they provide an explanation for the prediction of a model without knowing the internal mechanism of the model. However, there is always a gap between the their explanations and how the model truly “reasons”. This problem does not exist for white-box classifiers, i.e., logistic regression and pattern aided techniques, which are intrinsically interpretable and from which explanations can be directly extracted. The main limitation with interpretable models is that they may fail to achieve competitive accuracy with more complex models. Our method is an interpretable model that is designed to improve accuracy via enriching the model complexity by forming a $2^{|P|}$ -hypothesis space from $|P|$ partitions.

7.3 Preliminaries

Expert explanation.

Given a D -dimensional instance x , an expert explanation for x is represented by a D -dimensional binary vector $(e_1, \dots, e_D)$ where e_i is 1 if the expert considers it’s reasonable to use the i th feature when inferring the class label, and 0 if the expert considers it should not be used when inferring the label. Note that different experts might produce different explanations about an instance, according to different logic. We treat the generating process behind an expert explanation as a black box.

Model Explanation.

Given a D -dimensional instance x and a model f with classification decision $f(x)$, the model explanation is a D -dimensional vector $(e_1, e_2, \dots, e_D)$ where e_i is 1 if the i th feature is used by the model to make the classification decision $f(x)$ and 0 otherwise.

We assume there is some agreed procedure for determining when a feature is used by the model to make the classification decision for white-box classifiers. For example, the features used for classifying x in a decision tree are the 1's in the explanation. For a black box classifier, the explanations can be extracted with an explanation system like LIME. The non-zero coefficients of a regression formula are the 1's in the explanation for all instances.

Reliability.

Reliability is one important property of the explanations and it measures the closeness of the logical reasoning behind a model to the expert knowledge. There are many possibilities such as Dice, Jaccard similarity or F_1 score to measure it and we adopt the F_1 score (which is used in (Ribeiro, Singh, and Guestrin, 2016c)) to quantify the reliability. Given an expert explanation e_{true} , a D -dimensional instance x and a model f whose explanation for $f(x)$ is e , the reliability of the model's prediction is measured by the F_1 score: $F_1 = 2 \frac{precision * recall}{precision + recall}$, where $precision(e, e_{true}) = \frac{\sum_{d \in D} |e_d \times e_{true,d}|}{\sum_{d \in D} |e_d|}$ represents the fraction of features used by the model that are expert agreed, and $recall(e, e_{true}) = \frac{\sum_{d \in D} |e_d \times e_{true,d}|}{\sum_{d \in D} |e_{true,d}|}$ represents the fraction of expert-agreed features that the model covers.

7.4 Methodology

7.4.1 Model description

Given a pattern set $P = \{p_1, p_2, \dots, p_K\}$, the overall predictor is defined as:

$$f(x) = \sum_{p \in P} \lambda_p(x) \sum_{d=0}^D a_{dp} x_d \quad (7.1)$$

where,

- $\lambda_p(x) \in \{0, 1\}$ is the activation function for x given pattern p , and is 1 if x matches p .
- D is the number of features in each instance.
- a_p is a $(D + 1)$ -dimensional vector of coefficients associated with pattern p , where a_{dp} corresponds to d -th attribute and a_{0p} is the bias term.

Moreover, $f(x)$ can be viewed as a combined linear model $f(x) = \sum_{d=1}^D \omega_d x_d + \omega_0$, where the d -th coefficient ω_d is $\omega_d = \sum_p \lambda_p(x) a_{dp}$ and the bias $\omega_0 = \sum_p \lambda_p(x) a_{0p}$. One advantage of partition-wise linear models is that $|P|$ vectors of coefficients can generate $2^{|P|}$ possible linear models.

In particular, this chapter explores xPWL for binary classification such that the class label $c \in \{0, 1\}$, and xPWL returns 1 if $f(x) \geq 0$ and 0 otherwise. It can be extended to multi-class classification in the same manner as logistic regression, and xPWL can also be applied for regression.

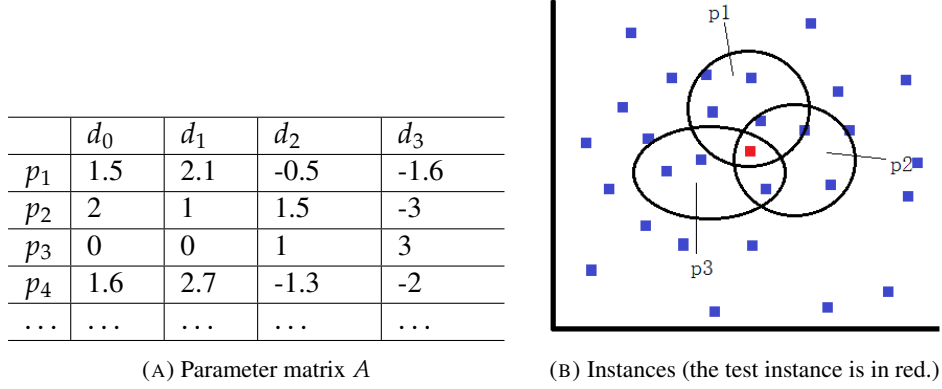


FIGURE 7.1: Demonstration of xPWL

An example of xPWL is shown in Figure 7.1, K patterns are mined from the training data and every pattern is associated with a linear model represented by a vector of weights (shown in Figure 7.1a). Suppose that a test instance x (red point in Figure 7.1b) matches patterns p_1, p_2 and p_3 , and these patterns are assigned with vector $(1.5, 2.1, -0.5, -1.6)$, $(2, 1, 1.5, -3)$ and $(0, 0, 1, 3)$ respectively. The instance-specific predictor for x is the linear combination of the weights of matching patterns p_1, p_2 and p_3 , and the coefficient vector is $(1.5, 2.1, -0.5, -1.6) + (2, 1, 1.5, -3) + (0, 0, 1, 3) = (3.5, 3.1, 0, -1.6)$.

7.4.2 Model explanations of xPWL

δ -importance. Given a D -dimensional instance x and an interpretable linear model $f(x) = \sum_{d=1}^D \omega_d x_d + \omega_0$, the d -th feature of x is considered δ -important for the prediction of x by f if the d -th coefficient $|\omega_d| \geq \delta$.

Explanation of xPWL. Given a D -dimensional instance x and an interpretable linear model $f(x) = \sum_{d=1}^D \omega_d x_d + \omega_0$, the explanation for the prediction of x by f can be represented by a D -dimensional vector $(e_1, e_2, \dots, e_D)$, where $e_d = 1$ if the d -th feature of x is δ -important. Formally, a function $g(x|f)$ is denoted to give the explanation for x :

$$g(x|f) = (g_1(x|f), g_2(x|f), \dots, g_D(x|f)) \quad (7.2)$$

where $g_d(x|f) = \mathbb{1}_{[\sum_p \lambda_p(x) a_{dp} \geq \delta]}$, and $\mathbb{1}_{[C]}$ is an indicator function that returns 1/0 if the condition C is true/false.

7.4.3 Objective function

The objective function for the learning process is defined as:

$$\begin{aligned} \min_A \quad & \frac{1}{N} \sum_n l_{acc}(y_n, f(x_n)) + \alpha \frac{1}{M} \sum_m l_{exp}(e_{x_m}, g(x_m)) \\ \text{subject to} \quad & \sum_{p \in P} \sum_{d=1}^D \mathbb{1}_{[a_{dp} \neq 0]} \leq \mu, \end{aligned} \quad (7.3)$$

where $A = (a_1, \dots, a_{|P|})$ is the parameter matrix of xPWL, N is the number of training instances, M is the number of training instances whose explanations are provided, and α is the reliability regularization rate. In the objective function, $l_{acc}(y_n, f(x_n))$ denotes the loss function in terms of accuracy where y_n is the true prediction for x_n , and

$l_{exp}(e_{x_m}, g(x_m))$ denotes the loss function in terms of explanation quality where e_{x_m} is the expert explanation for x_m . The sparsity constraint enforces the requirement that only a small number of entries in the patterns/partitions are active to mitigate the effect of overfitting.

7.4.4 Loss functions

In (7.3) the objective function consists of two parts: an accuracy loss function l_{acc} and an reliability loss function l_{exp} .

Accuracy loss function l_{acc} The loss function in terms of accuracy can be chosen according to the specific application or domain. For classification, squared hinge or logistic loss can be used. For regression, squared loss can be used. Logistic loss function is used as an accuracy loss function for classification in this chapter.

Reliability loss function l_{exp} Given an expert explanation e_x and a model explanation $g(x|f)$ (defined in (7.2)) for an instance x and the predictor f , the loss function in terms of the reliability is defined as

$$\begin{aligned} l_{exp}(e_x, g(x)) &= \sum_{d=1}^D (g_d(x|f) - e_{x,d})^2 \\ &= \sum_{d=1}^D (\mathbb{1}_{[\sum_p \lambda_p(x) a_{dp} \geq \delta]} - e_{x,d})^2 \\ &= \sum_{d=1}^D ((1 - 2e_{x,d}) \mathbb{1}_{[\sum_p \lambda_p(x) a_{dp} \geq \delta]} + e_{x,d}^2) \end{aligned} \quad (7.4)$$

where δ denotes δ -importance, and it is a small value that is usually close to 0 (we use 0.001 in our setting, the experiments suggest that the performance is stable when $\delta < 10^{-2}$). Because the last term $e_{x,d}^2$ is constant, we can drop it from in the loss function. Finally, the reliability loss l_{exp} is derived as:

$$l_{exp}(e_x, g(x)) = \sum_{d=1}^D (1 - 2e_{x,d}) \mathbb{1}_{[\sum_p \lambda_p(x) a_{dp} \geq \delta]} \quad (7.5)$$

The reliability loss function in (7.5) is non-convex, which raises challenges for optimizing the objective function. We employ a convex upper bound as the new loss functions:

$$\begin{aligned} l_{exp}(e_x, g(x)) &= \sum_{d=1}^D (1 - 2e_{x,d}) \mathbb{1}_{[\sum_p \lambda_p(x) a_{dp} \geq \delta]} \\ &\leq \sum_{d=1}^D (1 - 2e_{x,d}) (|\sum_p \lambda_p(x) a_{dp}| - \delta + 1) \end{aligned} \quad (7.6)$$

Also, the constant term $\delta - 1$ can be removed for optimization purposes. Then the final reliability loss function is:

$$l_{exp}(e_x, g(x)) = \sum_{d=1}^D (1 - 2e_{x,d}) |\sum_p \lambda_p(x) a_{dp}| \quad (7.7)$$

There also exist other possibilities to define the reliability loss function, i.e., a logistic loss like function. The one used in (7.6) has shown promising results in our experiments, and it will be left as future work to explore other possibilities.

7.4.5 Convex relaxation

We consider the convex relaxation of (7.3) by a standard approach to adopt a L_1 constraint that is convex and differentiable to replace the original L_0 constraint. The new objective function becomes:

$$\min_A \mathcal{J}(A), \quad \text{subject to} \quad \sum_{p \in P} \sum_{d=1}^D |a_{dp}| \leq \mu \quad (7.8)$$

where $\mathcal{J}(A) = \frac{1}{N} \sum_n l_{acc}(y_n, f(x_n)) + \alpha \frac{1}{M} \sum_m \sum_{d=1}^D (1 - 2e_{x_m,d}) |\sum_p \lambda_p(x_m) a_{dp}|$, A is the parameter matrix. Then we can reformulate (7.8) to the equivalent problem as follows:

$$\min_A \mathcal{J}(A) + \beta \sum_{p \in P} \sum_{d=1}^D |a_{dp}| \quad (7.9)$$

where β is the regularization weight corresponding to μ .

7.4.6 Solution

This optimization can be solved via the proximal method (Parikh and Boyd, 2014), which iteratively applies gradient steps and proximal steps to update parameters.

Assume that $A^{(t)}$ represents the parameter matrix $A = (a_0, \dots, a_{|P|})$ at the t -th iteration. First, in the gradient step, $A^{(t)}$ is updated according to the gradient of $\mathcal{J}$. Note that the accuracy loss function we used is the logistic loss function that is differentiable, and for the reliability loss function, we approximate the gradient by making use of the formula $\partial_x |x| = \text{sign}(x)$ when $x \neq 0$, where $\text{sign}(x)$ returns 1 if x is positive and -1 otherwise. The gradient of the reliability loss function is then

$$\partial_{a_{dp}} \Omega(A) = \alpha \frac{1}{M} \sum_m \text{sign}(\sum_p \lambda_p(x_m) a_{dp}) (1 - 2e_{x_m,d}) \lambda_p(x_m)$$

where $\Omega(A) = \alpha \frac{1}{M} \sum_m \sum_{d=1}^D (1 - 2e_{x_m,d}) |\sum_p \lambda_p(x_m) a_{dp}|$. Formally, the update for gradient step is:

$$A^{(t+\frac{1}{2})} = A^{(t)} - \eta^{(t)} \partial_{A^{(t)}} \mathcal{J}(A) \quad (7.10)$$

where $\eta^{(t)}$ is the step size, and $\partial_{A^{(t)}}$ the 1st-order derivative.

In the proximal step, parameters are updated by solving:

$$A^{(t+1)} = M_0(A^{(t+\frac{1}{2})}) \quad (7.11)$$

where $M_0(B) = \underset{A}{\operatorname{argmin}} (\frac{1}{2} \|A - B\|^2 + \eta^{(t)} \beta \sum_{p \in P} \sum_{d=1}^D |a_{dp}|)$.

$M_0(B)$ is a well-known soft-thresholding operator, and can be decomposed into an element-wise problem (Oiwa and Fujimaki, 2014), and the update formula is as follows:

$$a_{dp} = \begin{cases} b_{dp} - \eta^{(t)} \beta, & b_{dp} > \eta^{(t)} \beta \\ 0, & |b_{dp}| \leq \eta^{(t)} \beta \\ b_{dp} + \eta^{(t)} \beta, & b_{dp} < -\eta^{(t)} \beta \end{cases}$$

Algorithm 10: Learning process of xPWL

Input: training data $X = \{x_1, x_2, \dots, x_N\}$, explanations $E = \{e_1, e_2, \dots, e_M\}$,
 $minSupp$
Output: P, A

```

1  $P = \text{minePatterns}(X, minSupp)$ ;
2 Add a global pattern  $p_0$  into  $P$ ;
3 Initialize  $A^{(0)}$  with 0s;
4  $t = 1$ ;
5 while  $t < MaxIteration$  do
6   /*gradient step*/ ;
7   Derive  $A^{(t+\frac{1}{2})}$  by calculating (7.10);
8   /*proximal step*/ ;
9   Derive  $A^{(t+1)}$  by calculating (7.11);
10  if termination condition is true then
11    break;
12  end
13 end
14 return  $P, A$ ;
```

7.5 Algorithm

We describe the proposed explanation-based partition-wise linear models (xPWL) in this section. First, a set of contrast patterns are mined from the training data. Then the vectors of weights associated with the patterns are iteratively updated as discussed. The framework of the algorithm is shown in Algorithm 10 and the details are also discussed.

Inputs. The input parameters of the learning process are:

- Training data $X = \{x_1, x_2, \dots, x_N\}$ contains the labeled instances to train the model;
- Expert explanations $E = \{e_1, e_2, \dots, e_M\}$, where $M \leq N$, is the expert supervision provided to train the model. Every explanation in E corresponds to an instance in X , but not all training instances are necessarily associated with explanations. Such expert supervision could be collected by the experts when labeling the data. We assume such information is given.
- $minSupp$ is the essential parameter for pattern mining. The choice of small values results in a large number of patterns that are computationally difficult to deal with. On the other hand, the choice of large values may fail to find representative patterns. The tree based mining algorithm (Shang et al., 2016) used in our experiments does not need the user to specify the $minRatio$.

Pattern mining. The first step is to generate a contrast pattern set $P = \{p_1, \dots, p_K\}$ from the data to represent partitions. Experimental results in (Dong and Taslimitehrani, 2016) have shown to be promising for the use of contrast patterns as local regions. One advantage of contrast patterns is that they are able to find discriminative feature combinations compared with traditional frequent patterns. Another advantage is that contrast patterns are intrinsically interpretable compared with other partitioning techniques such as spectral clustering (Arias-Castro, Chen, and Lerman, 2011) and parallel coordination division (Oiwa and Fujimaki, 2014). We adopt a random forest based pattern mining method (Shang et al., 2016), which does not require user-specified parameter $minRatio$, to generate contrast patterns for its efficiency. There also exist many mining techniques

for contrast pattern mining, such as GcGrowth (Li et al., 2005), subgroup discovery (Atzmueller, 2015) and exceptional model mining (Duivesteijn, Feelders, and Knobbe, 2016). Interested users can refer to (Dong and Bailey, 2012) for further discussion about contrast patterns. After the pattern set P is mined from the training data, every pattern/partition is associated with a $(D + 1)$ dimensional vector of weights, and these vectors of weights are represented as a $|P| \times (|D| + 1)$ matrix A , where a_{dp} is the d th weight associated with pattern p . It should be pointed out that a partition can overlap with other partitions, and the pattern set may not cover the whole sample space. Therefore, we always add a global pattern, which is a synthetic pattern such that every instance matches it, to the pattern set. The linear model associated with the global pattern can be treated as the global model, and the linear models associated with other patterns can be treated as local residual models.

Parameter learning process. The task of learning the model is to find a parameter matrix A^* to optimize the objective function described in (7.8). A proximal method is adopted to iteratively update element-wise $a_{dp}, \forall d = \{0, 1, 2, \dots, D\}, p \in P$. The termination condition is that the differences in the objective values between the current iteration and the previous one are less than 10^{-9} in 10 consecutive iterations, as used in PWL. There are many techniques (Ruder, 2016) to adjust the step size for quick converges.

Time complexity analysis. The learning process is mainly pattern mining and parameter update. The time complexity of the tree based pattern mining method is $O(RlE)$, where R is the number of trees, l is the number of instances on each level of a tree and E is the tree depth threshold. At each iteration of the update, all the training instances are processed to calculate the gradients, and every entry of A is updated once. Thus the time complexity for the update process is $O(T(N + |P|D))$, where T is the number of iterations, $|P|$ is the number of patterns and D is the number of features. The overall time complexity is then $O(RlE + T(N + |P|D))$.

7.6 Experiments

We evaluate the proposed method xPWL against several baselines in this section. We demonstrate that how xPWL is able to use extra information to improve the reliability of the predictions (which is similar to use user-specified examples as side information to improve the clustering performance in (Xing et al., 2003) in the sense that we). The proposed method is compared with baselines that achieve high accuracy but are not able to incorporate extra information directly in the training process. The experiments are conducted on UCI datasets, and on two scenarios where the expert explanations arise in different ways. The sensitivity to the setting of the main parameters is also investigated.

7.6.1 Settings

Data and pre-processing. Twelve UCI datasets (adult, balloon, blood, crx, diabetes, hepatitis, hypo, labor, musk, sick, titanic, vote) described in Table 8.3 are used in this section, and each data set is randomly split into two groups such that 80% is used to train the classifier and 20% is used for testing. Non-numeric features are binarized using one-hot encoding method, which are also treated as numeric features, then all features are normalized into the range of $[0, 1]$ using feature scaling.

Experiments settings. Two major model parameters η (step size) and β (sparsity regularization rate) are set via a nested validation process. The training data are divided into 80% and 20%, where the latter 20% are used for validation to choose the settings with the best performance. Then the chosen setting is applied on the whole training data to train the model. The reliability regularization rate α is set to 0.5, and $minSupp=0.1$, and the impacts of these two parameters are discussed in later subsections as well.

Dataset	#instances	#nominal_attr	#numeric_attr	#classes
Adult	48842	8	6	2
Balloon	16	4	0	2
Blood	758	0	4	2
Crx	690	9	6	2
Diabetes	768	0	8	2
Hepatitis	155	188	1	2
Hypo	2800	1	24	2
Labor	57	8	8	2
Musk	6598	0	168	2
Sick	3772	22	7	2
Titanic	3772	22	7	2
Vote	435	16	0	2

TABLE 7.2: Data description

The metrics including both the accuracy and explanation reliability are evaluated using 10-fold cross validation.

Benchmark methods. We evaluate the following methods: **xPWL** - Explanation-based partition-wise linear models; **PWL** - Partition-wise linear models; **LR** - Logistic regression; **LR-PCA** - Logistic regression with PCA pre-processing; **MLP** - Multilayer perception; **RF** - Random forest. For black-box classifiers MLP and RF, the explanations are provided by LIME. As the traditional methods are not able to work with the expert explanations, we also evaluate the baselines in a way such that the expert explanations can be used. As used in (Ribeiro, Singh, and Guestrin, 2016), a pre-processing procedure, which deletes the features (sets the feature values as missing) that are not considered as important in the expert explanation for a training instance, can be employed for traditional classifiers to utilize the expert explanations. The baselines with such procedure are denoted by the “*” symbol (e.g., RF* denotes that a random forest is trained with the expert explanations).

7.6.2 Evaluation

Experiments on explanations from decision trees. In this test, we evaluate the performance of the proposed method with benchmark methods on datasets where the expert explanations are extracted from decision trees. For each UCI dataset, a decision tree is trained on the dataset and then all instances are re-labeled using the decision tree, together with the explanations for the instances. Note that a feature is considered important if it occurs in the prediction path from the root to the leaf of the decision tree given an instance, and expert explanations for all training instances are generated.

The accuracy results are shown in Table 8.6 and reliability results are shown in Table 8.5. It can be seen that xPWL achieves a large improvement in terms of reliability compared with the baselines, as it wins in all tested datasets. In terms of accuracy, xPWL is still able to achieve competitive performance. Therefore the experimental results suggest that xPWL is able to make accurate and reliable predictions.

Experiments on UCI data with noisy features. In this test, we randomly add 50% noisy numeric features to each original UCI dataset, then treat the original features as expert explanations. For example, the Balloon dataset contains four features - *age*, *act*, *color*, *size*, and we add two numeric features - n_1, n_2 with randomly assigned values, then the

Accuracy(%)												
	Proposed	Baselines without expert explanations					Baselines with expert explanations					
Dataset	xPWL	PWL	LR	LR-PCA	MLP	RF	PWL*	LR*	LR-PCA*	MLP*	RF*	
Adult	95.24	93.66	92.54	94.57	93.28	96.67	76.56	74.60	79.24	74.89	81.22	
Balloon	100.00	100.00	100.00	100.00	100.00	100.00	50.00	100.00	75.00	100.00	100.00	
Blood	98.00	90.67	84.00	84.00	86.00	100.00	52.67	54.00	82.67	74.67	100.00	
Crx	91.67	93.05	93.05	88.43	91.25	93.05	87.43	84.54	85.54	87.42	89.20	
Diabetes	88.31	74.68	84.42	88.31	85.06	92.21	83.12	71.43	64.29	67.53	91.56	
Hepatitis	90.32	90.32	96.77	96.77	93.54	100.00	100.00	80.65	87.10	51.61	100.00	
Hypo	97.84	92.39	97.22	92.39	96.50	99.69	53.09	72.33	91.26	27.88	97.84	
Labor	91.67	83.33	83.33	75.00	66.67	83.33	66.67	66.67	66.67	66.67	91.67	
Musk	95.13	88.38	92.13	93.23	93.24	96.68	76.48	75.54	73.23	71.57	78.93	
Sick	96.07	95.36	95.10	96.12	98.01	97.79	88.13	88.13	83.40	91.04	88.73	
Titanic	91.87	88.04	87.80	79.90	89.23	97.84	83.49	72.97	65.07	63.64	97.61	
Vote	93.10	93.10	97.70	90.80	91.95	98.85	97.70	91.95	58.62	93.10	100.00	

TABLE 7.3: Accuracy results where expert explanations come from decision trees (with the best values in **bold**). xPWL - our proposed method; PWL - Partition-wise linear models; LR - Logistic regression; LR-PCA - LR with PCA pre-processing; MLP - Multilayer perception; RF - Random forest. “**” denotes that the corresponding method is trained with expert explanations

expert explanation for all instances is $(1, 1, 1, 1, 0, 0)$, where the 1s represent the original features. Full expert explanations for training data are used.

The accuracy results are shown in Table 8.8 and reliability results are shown in Table 8.7. For the case where expert explanations consist of non-noisy features, xPWL achieves competitive performance in terms of accuracy. Moreover, xPWL wins in 10 out of 12 datasets in terms of reliability. It is worth pointing out that though PCA is known to behave well in datasets with noisy features, xPWL is still able to outperform PCA.

Impact of α . α manages the balance between accuracy and reliability. The evaluations when varying α are shown in Figure 7.2. The results suggest that the accuracy slowly decreases as α increases and becomes stable or better when α is greater than 10, while the reliability increases as α increases until $\alpha = 10$ and decreases when $\alpha > 10$. In general, it is good to set α in the range of $[0, 1]$, and it achieves the best results on average when $\alpha = 0.5$.

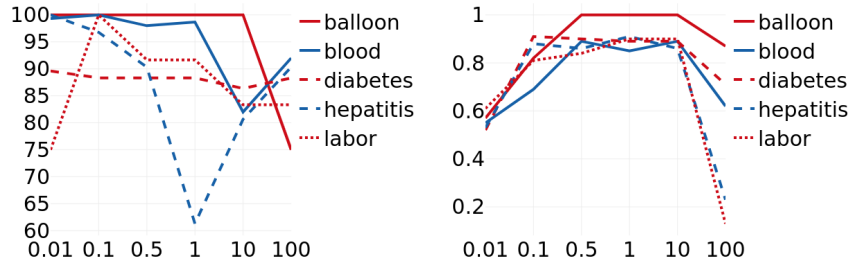


FIGURE 7.2: Impact of α . X-axis represents α and y-axis represents accuracy (left) and reliability (right).

Impact of $minSupp$. In this part, we investigate the impact of the main parameter in pattern based methods: $minSupp$. The impact of $minSupp$ on accuracy and reliability is shown in Figure 7.3, where the overall performance in terms of both accuracy and reliability is stable at first (the value of $minSupp$ depends on the specific dataset) and then decreases with $minSupp$ increasing. The results suggest that it is reasonable to set $minSupp$ in the range of $[0.1, 0.2]$.

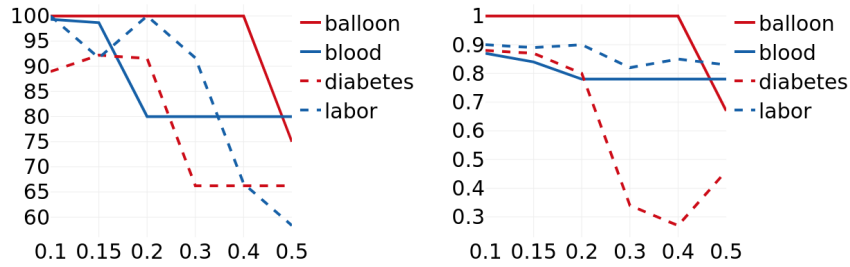


FIGURE 7.3: Impact of size of $minSupp$. X-axis represents $minSupp$ and y-axis represents accuracy (left) and reliability (right).

Impact of level of expert supervision. The performance results (in the scenario where the expert explanations are from decision trees, and the trend is similar in the other scenario) when varying the level of expert supervision from 0% to 100% with steps of 20% are reported in Figure 7.4, i.e., 20% of expert supervision denotes that 20% of the training instances are provided with expert explanations for learning. The results suggest that the accuracy is stable when varying the level of expert supervision. The reliability increases and becomes stable when the expert supervision is 20% and above, which suggests that

providing a small fraction (20% in this case) of the training instances with expert supervision is sufficient to train xPWL to make reliable predictions.

Accuracy and reliability. It can be seen from the experimental results that the accuracy and the reliability do not necessarily contradict with each other. xPWL is able to improve the reliability using expert supervision without jeopardizing the accuracy, while traditional classifiers focus solely on accuracy, and are not able to work with such information during the training process.

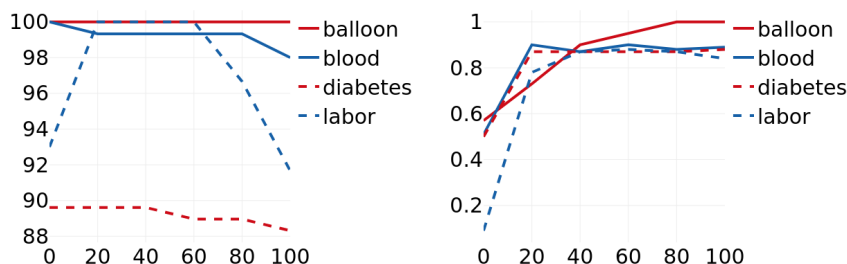


FIGURE 7.4: Impact of level of expert supervision. X-axis represents percentage of expert supervision and y-axis represents accuracy (left) and reliability (right).

7.7 Conclusion

This chapter proposes explanation-based partition-wise linear models (xPWL) that improve the ability to involve explanations in the training process, in order to make reliable predictions. By taking a small level of expert supervision as extra information, xPWL is able to achieve high reliability as well as high accuracy. The experimental results show that with less than 20% of expert supervision, xPWL is able to consistently achieve more than 80% reliability, while the accuracy is as good as the baselines. xPWL is a new promising interpretable model for scenarios (e.g., medical diagnosis, criminal analysis and marketing) where interpretability is desired (explanations are required), and high reliability of explanations is as important as high accuracy. We also expect that the effectiveness of this type of model will encourage the collection of data explanations when possible.

The mainly limitation of xPWL is that while the expert explanation may increase the reliability of the prediction, it also may constrain the model towards “well-known” rules, making more difficult the discovery of new relationships among variables.

Reliability												
	Proposed		Baselines without expert explanations					Baselines with expert explanations				
Dataset	xPWL	PWL	LR	LR-PCA	MLP	RF	PWL*	LR*	LR-PCA*	MLP*	RF*	
Adult	0.92	0.58	0.48	0.55	0.61	0.67	0.54	0.51	0.53	0.63	0.71	
Balloon	1.00	0.68	1.00	0.67	0.67	0.67	0.60	0.00	0.66	0.67	0.67	
Blood	0.89	0.59	0.59	0.71	0.83	0.89	0.51	0.62	0.71	0.78	0.89	
Crx	0.91	0.63	0.34	0.39	0.41	0.53	0.59	0.34	0.38	0.48	0.53	
Diabetes	0.88	0.63	0.58	0.35	0.60	0.66	0.54	0.49	0.63	0.35	0.69	
Hepatitis	0.86	0.22	0.23	0.17	0.25	0.08	0.35	0.11	0.24	0.04	0.61	
Hypo	0.93	0.10	0.48	0.01	0.48	0.01	0.25	0.00	0.01	0.14	0.18	
Labor	0.84	0.17	0.20	0.28	0.03	0.29	0.69	0.55	0.16	0.43	0.75	
Musk	0.93	0.37	0.46	0.47	0.68	0.52	0.34	0.37	0.42	0.63	0.52	
Sick	0.85	0.45	0.49	0.51	0.59	0.33	0.43	0.49	0.54	0.59	0.75	
Titanic	0.81	0.48	0.58	0.35	0.12	0.56	0.47	0.39	0.44	0.20	0.72	
Vote	0.95	0.19	0.35	0.17	0.10	0.61	0.17	0.39	0.20	0.34	0.74	

TABLE 7.4: Reliability results where expert explanations come from decision trees (with the best values in **bold**). xPWL - our proposed method; PWL - Partition-wise linear models; LR - Logistic regression; LR-PCA - LR with PCA pre-processing; MLP - Multilayer perceptron; RF - Random forest. “*” denotes that the corresponding method is trained with expert explanations

Accuracy(%)												
	Proposed	Baselines without expert explanations					Baselines with expert explanations					
Dataset	xPWL	PWL	LR	LR-PCA	MLP	RF	PWL*	LR*	LR-PCA*	MLP*	RF*	
Adult	84.43	84.13	78.92	85.76	88.12	88.67	81.82	78.92	83.23	84.63	81.22	
Balloon	100.00	100.00	100.00	100.00	100.00	100.00	25.00	100.00	100.00	100.00	100.00	
Blood	78.67	81.33	80.67	80.67	78.67	76.67	78.00	82.00	80.66	78.67	68.67	
Crx	77.50	75.02	69.45	73.42	75.02	79.80	59.00	62.33	72.24	69.32	71.24	
Diabetes	69.48	68.83	75.32	74.03	75.32	70.13	72.73	74.03	69.48	74.68	74.68	
Hepatitis	74.19	83.87	74.19	80.65	77.42	74.19	80.65	77.42	83.87	77.42	80.65	
Hypo	96.30	92.70	97.22	92.70	96.60	99.38	93.00	97.63	92.70	96.40	99.59	
Labor	91.67	91.67	91.67	91.67	91.67	58.33	58.33	91.67	83.33	91.67	58.33	
Musk	81.37	76.23	68.89	78.85	71.37	79.24	78.90	72.29	80.81	72.49	82.33	
Sick	93.33	92.10	88.49	88.49	87.42	91.20	78.24	81.50	83.83	76.57	74.24	
Titanic	23.92	23.92	38.28	25.84	27.27	30.62	16.27	37.32	39.00	27.51	32.54	
Vote	94.25	95.40	97.70	90.06	91.95	100.00	90.80	98.85	90.80	91.95	97.70	

TABLE 7.5: Accuracy results where expert explanations come from non-noisy features (with the best values in **bold**). xPWL - our proposed method; PWL - Partition-wise linear models; LR - Logistic regression; LR-PCA - LR with PCA pre-processing; MLP - Multilayer perceptron; RF - Random forest. “*” denotes that the corresponding method is trained with expert explanations

Reliability											
	Proposed	Baselines without expert explanations					Baselines with expert explanations				
Dataset	xPWL	PWL	LR	LR-PCA	MLP	RF	PWL*	LR*	LR-PCA*	MLP*	RF*
Adult	0.85	0.49	0.58	0.65	0.53	0.71	0.61	0.68	0.71	0.61	0.73
Balloon	0.96	0.78	0.57	0.89	0.72	0.40	0.48	0.67	0.80	0.40	0.40
Blood	0.98	0.73	0.91	0.80	0.70	0.54	0.89	1.00	0.80	0.71	0.61
Crx	0.72	0.23	0.14	0.17	0.32	0.42	0.23	0.19	0.28	0.36	0.42
Diabetes	0.83	0.74	0.75	0.80	0.77	0.40	0.94	0.86	0.80	0.58	0.43
Hepatitis	0.74	0.58	0.40	0.50	0.32	0.01	0.64	0.35	0.60	0.10	0.01
Hypo	0.64	0.57	0.17	0.36	0.11	0.00	0.29	0.11	0.54	0.04	0.00
Labor	0.75	0.68	0.38	0.59	0.11	0.01	0.52	0.47	0.65	0.02	0.01
Musk	0.82	0.57	0.59	0.48	0.42	0.61	0.60	0.71	0.51	0.42	0.69
Sick	0.91	0.65	0.48	0.43	0.52	0.49	0.64	0.59	0.53	0.56	0.67
Titanic	0.45	0.32	0.27	0.22	0.20	0.20	0.27	0.37	0.19	0.27	0.21
Vote	0.90	0.86	0.43	0.43	0.34	0.09	0.88	0.61	0.41	0.05	0.09

TABLE 7.6: Reliability results where expert explanations come from non-noisy features (with the best values in **bold**). xPWL - our proposed method; PWL - Partition-wise linear models; LR - Logistic regression; LR-PCA - LR with PCA pre-processing; MLP - Multilayer perception; RF - Random forest. “*” denotes that the corresponding method is trained with expert explanations

Chapter 8

Joint Learning for Classification and Pattern Based Explanations

8.1 Introduction

Traditional classifiers such as decision trees and linear models that are of good interpretability often suffer from low prediction accuracy. On the contrary, modern classifiers such as SVMs and deep neural networks (DNNs) that have good prediction performance are often difficult for humans to understand the reasoning logic associated with their predictions. Therefore, techniques that can address both accuracy and interpretability are critical for advanced and interpretable machine learning. In this chapter, we propose a neural network based framework (named ExplNet) to simultaneously learn classification and explanations. ExplNet is a multi-task learning framework that is trained jointly with class labels (for classification) and pattern-based explanations (for interpretability) with two different subnetworks on top of a shared base network. The explanation subnetwork provides explanations for the network's prediction at the classification subnetwork. Contrast patterns are used to train the explanation subnetwork to output a human-understandable logical pattern at each of its output unit. We empirically show, on a range of benchmark datasets, that ExplNet is able to achieve superior interpretability with comparable accuracy to classification-only classifiers. We also demonstrate that contrast patterns are a good choice for explanations when ground truth explanations are not available, a common scenario for real-world applications.

The lack of interpretability for state-of-the-art black-box classifiers is an important issue for their deployment. Traditional classifiers suffer from a trade-off between accuracy and interpretability, such that interpretable models (e.g., decision trees, linear models) usually achieve poor accuracy, while black-box models (e.g., SVMs, neural networks) make it difficult for humans to understand their logical reasoning. Thus, we require techniques that can improve interpretability without sacrificing accuracy. In this chapter, we propose a neural network (called ExplNet) based on a multi-task learning framework that is trained jointly with class labels and explanation units, and is able to provide explanations as outputs. ExplNet has two separate layers (explanation layer and label layer) following the shared layers, and the loss function is a weighted summation over the loss functions for the two layers. The outputs for the explanation layer are the explanation units where every unit represents a human-understandable logical pattern. Due to the difficulty of collecting the explanation units in practice, we also propose a method that uses contrast patterns as the estimation of the explanation units. Our experiments show that ExplNet is able to achieve superior interpretability and comparable accuracy when compared with several baseline classifiers, and that contrast patterns are a good choice for explanation units when ground truth explanations are not available.

Recently, several techniques have been proposed to improve the interpretability of black-box classifiers by extracting model-agnostic explanations (Ribeiro, Singh, and Guestrin,

(2016b) (Henelius et al., 2014) (Robnik-Šikonja and Kononenko, 2008) (Ribeiro, Singh, and Guestrin, 2016c) (Ribeiro, Singh, and Guestrin, 2018). With these extracted explanations, human decision-makers can gain insight into the classifier, particularly in regard to feature importance and how the predictions are made. These explanations are often compared with expert knowledge or some ground truth, and we call a prediction to be *reliable* if its explanation agrees with the expert ground truth. The reliability of a prediction can be as important as its accuracy, especially in domains where decisions that are based on the predictions might have severe consequences, for instance, in medical analysis and criminal investigations.

	High Accuracy	Low Accuracy
High Reliability	Users are most satisfied.	The logic of the model might be wrong.
Low Reliability	The prediction could be untrustworthy.	Model's prediction is not useful.

TABLE 8.1: Four scenarios on how reliability and accuracy of a prediction could influence a decision making process.

In Table 8.1, we list four possible scenarios on how the reliability and accuracy of a prediction could influence a decision making process. It is desirable that a prediction is both accurate and reliable, that is, the prediction is correct and its explanation matches the domain knowledge. If an explanation is of poor quality, then the prediction is no better than a random guess as the logical reasoning behind the model could be misinterpreted. A toy dataset that can help explain this point in more detail can be found in Table 8.2. There are three features: person (who is blowing up a balloon), color (the color of the balloon), pressure (how much pressure is used to blow up the balloon), and two classes indicating whether the balloon will “burst” or not. According to human knowledge, pressure is probably the most reasonable explanation for the future status of a balloon, whilst color and person may be irrelevant features. In practice, a classifier might conceivably make its decisions based on the “person”, which somehow relates to the “pressure” (e.g., a person who has large lung capacity can produce large pressure). Then, the explanation for a test instance might be: “the balloon explodes because it is John who is blowing”, which is an unreliable prediction even though it is correct. In contrast, if another classifier learns to make decisions based on the “pressure”, then its predictions are reliable, since the explanation is consistent with the domain knowledge.

Person	Color	Pressure	Bursts
John	red	high	Y
John	blue	medium	Y
Jack	red	low	N
Jack	blue	medium	N
...	...	...	...

TABLE 8.2: A toy dataset about a person blowing a balloon.

Explanations improve model interpretability, but they are not directly addressed or integrated in standard machine learning models. Modern classifiers are often designed and trained specifically to make accurate predictions and not reliable/interpretable predictions. In this chapter, we attempt to close this gap, and answer the question: *how can one train a model to make both accurate and reliable predictions?* Our intuition is that this can be

addressed by multitask learning of two objectives, with one objective to learn a good classifier and the other objective to learn a good explainer. In this case of multi-task learning, the two objectives, i.e., learning explanations and learning class labels, are highly correlated. Therefore, we propose a neural network based approach with two subnetworks that learn jointly from a shared base network. One subnetwork is a classification subnetwork that takes inputs from the base network and outputs class probabilities, while the other subnetwork is an explanation network that also takes inputs from the base network but outputs their explanations. We denote our framework as ExplNet. Moreover, we propose to use patterns as explanations and further design the explanation subnetwork to output a human-understandable pattern at each of its output units.

Specifically, we propose an explanation loss function for explanation reliability, and combine it with the traditional classification loss to train the ExplNet model. The overall loss function of ExplNet is a Lagrange function of the two losses with a balancing Lagrange multiplier. The explanation loss can be regarded as a form of regularization that can help ease overfitting to class labels and focus more on interpretable patterns. To address the challenge that ground truth explanations are not always obtainable, we propose, alternatively, to use contrast patterns that are generated from the training data as explanations. Our main contributions are:

- We propose a novel neural network based framework, namely ExplNet, to learn simultaneously for classification and explanations in a multi-task learning scheme.
- ExplNet exploits contrast patterns as explanations for scenarios where expert knowledge is not available.
- We find that learning explanations and learning class labels are complementary tasks, and they can work jointly to improve each other under appropriate settings.
- We empirically show, on a range of benchmark datasets, that ExplNet can provide reasonable explanations and reliable predictions, with comparable accuracy to classification-only models.

8.2 Related Work

8.2.1 Multi-task learning

In machine learning, a typical classification task is to optimize a single classification objective, and the model is usually fine-tuned until the objective function reaches an optimum. During this learning process, useful information that helps better explain the model can be easily ignored, as such information may not be included explicitly in the classification objective but rather exists in some other related tasks. By sharing the basic representations between related tasks, multiple tasks can be jointly learned to achieve better performance than learn individually. These methods are called multi-task learning (MTL). Multi-task learning has been used successfully in many applications of machine learning, from natural language processing (Collobert and Weston, 2008), speech recognition (Deng, Hinton, and Kingsbury, 2013), computer vision (Salvador et al., 2016) and drug discovery (Ramsundar et al., 2015). MTL comes in many forms such as joint learning, learning to learn, and learning with auxiliary tasks. Generally, MTL is to optimize more than one loss functions at the same time, and we are effectively performing multi-task learning (in contrast to single-task learning). In this chapter, we will exploit multi-task learning to learn classification and explanations at the same time.

8.3 Our Proposed Framework - ExplNet

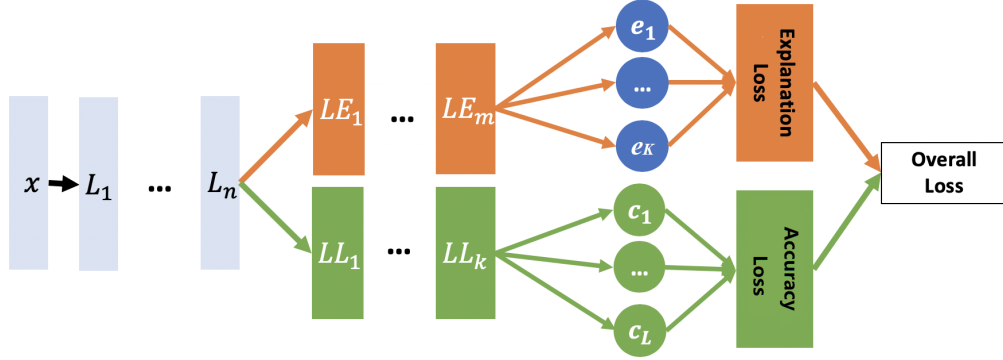


FIGURE 8.1: An overview of ExplNet structure. x is the input instance, $L_1, \dots, L_n$ are n shared fully connected layers, $LE_1, \dots, LE_m$ are m separate layers for explanation units $e_1, e_2, \dots, e_K$, and $LL_1, \dots, LL_k$ are k separate layers for class labels $c_1, \dots, c_L$

8.3.1 Explanation unit.

The explanation units for the prediction of an instance x is defined as a binary vector $(e_1, e_2, \dots, e_K)$, where $e_i \in \{0, 1\}$ and K is the number of explanations. Each unit represents a human-understandable explanation, and we adopt patterns as the representation of the explanation unit. Given a pattern set $\{p_1, p_2, \dots, p_K\}$, an explanation $(e_1, e_2, \dots, e_K)$ for a prediction means that pattern p_i is one of the logical explanations for the prediction if $e_i = 1$. If $e_i = 0$ then p_i is not an explanation for the prediction.

8.3.2 Overview

We employ hard parameter sharing Ruder, 2017 for multi-task learning (MTL) in ExplNet, which is one commonly used strategy for MTL with neural networks. The input is first fed into a shared base network with multiple hidden layers, then the outputs of the shared base network are passed into several task-specific subnetworks. The high level structure of ExplNet is shown in Figure 8.1. The input layer is followed by n fully connected shared layers $L_1, \dots, L_n$, then the last shared layer is connected to two separate subnetworks: $LE_1, \dots, LE_m$ for the explanation learning task, and $LL_1, \dots, LL_k$ for the classification learning task. The explanation units $e_1, e_2, \dots, e_K$ are the outputs of layer LE_m and the class labels (or probabilities) $c_1, \dots, c_L$ are the outputs of layer LL_k .

8.3.3 ExplNet

For ExplNet, we define the following notation: $n_{i,j}$ denotes the j th node in the L_i -th shared layer; $ne_{i,j}$ denotes the j th node in the explanation subnetwork layer LE_i ; $nl_{i,j}$ denotes the j th node in the classification subnetwork layer LL_i ; W denotes the weight matrix in the shared layers such that W_{ijo} denotes the weight for the connection from the j th node in L_i and the o th node in L_{i+1} , $i = 0, 1, \dots, n-1$ and L_0 is the input layer; U denotes the weight matrix in the explanation layers such that U_{ijo} denotes the weight for the connection from the j th node in LE_i and the o th node in LE_{i+1} , $i = 0, 1, \dots, m$, in particular $LE_0 = L_n$ is the last shared layer and LE_{m+1} is the explanation output layer; V denotes the weight matrix in the class learning layers such that V_{ijo} denotes the weight

for the connection from the j th node in LL_i and the o th node in LL_{i+1} , $i = 0, 1, \dots, k$, in particular $LL_0 = L_n$ is the last shared layer and LL_{k+1} is the class label output.

We use the sigmoid function as the activation function for all nodes (the exploration of different activation functions is out of the scope of this thesis, and the sigmoid function is sufficient to demonstrate the purpose of our method), such that the value of the o th node in the shared layer L_i is defined as:

$$y_{i,o} = (1 + e^{-\sum_j W_{i,j,o} * y_{i-1,j}})^{-1}, \quad (8.1)$$

where $\sum_j W_{i,j,o} * y_{i-1,j}$ is the weighed sum of the input connections from the previous layer. We can similarly define $ye_{i,o}$, the value of o th node in the explanation layer LE_i and $yl_{i,o}$, the value of the o th node in the classification label layer LL_i .

Given N input instances $X = (x_1, x_2, \dots, x_N)$ with true labels $T = (t_1, t_2, \dots, t_N)$ where $t_i = (t_{i,1}, t_{i,2}, \dots, t_{i,L})$, and expert explanations $S = (s_1, s_2, \dots, s_N)$ where $S_i = (s_{i,1}, s_{i,2}, \dots, s_{i,K})$, then the loss function for ExplNet is defined as:

$$L(X, T, S) = L_{acc}(X, T) + \alpha L_{expl}(X, S), \quad (8.2)$$

where L_{acc} is the loss function for the accuracy layer output $(c_1, \dots, c_L)$, L_{expl} is the loss function for explanation layer output $(e_1, e_2, \dots, e_K)$, and α is the Lagrange multiplier for L_{expl} . We adopt the most commonly used loss function, cross entropy, for the classification loss L_{acc} :

$$L_{acc}(X, T) = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^L t_{i,j} \log(p_{x_i, c_j}), \quad (8.3)$$

where p_{x_i, c_j} is the score on node c_j for x_i (the predicted probability that x_i is of class c_j).

The loss function for explanations is defined as the average binary cross entropy over all explanation units:

$$L_{expl}(X, S) = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^K (s_{i,j} \log(p_{x_i, e_j}) + (1 - s_{i,j}) \log(1 - p_{x_i, e_j})), \quad (8.4)$$

where p_{x_i, e_j} is the score on node e_j for x_i (the prediction probability that explanation unit e_j is activated).

The overall loss function is defined as:

$$L(X, S, E) = -\frac{1}{N} \sum_{i=1}^N \left(\sum_{j=1}^L t_{i,j} \log(p_{x_i, c_j}) + \alpha \sum_{j=1}^K (s_{i,j} \log(p_{x_i, e_j}) + (1 - s_{i,j}) \log(1 - p_{x_i, e_j})) \right) \quad (8.5)$$

Standard stochastic gradient descent (GSD) with Adam optimizer (Kingma and Ba, 2014) is applied to train ExplNet.

8.3.4 Contrast patterns as explanations

Ideally, the expert explanations for training should be provided by experts or extracted based on domain knowledge. However, due to the difficulty of collecting such information, there might be situations where ground truth expert explanations are not available. To

handle these situations, we propose to extract contrast patterns directly from the training data as the expert explanations.

The algorithm is described in Algorithm 11. It works with five parameters: (1) training instances $X = \{x_1, x_2, \dots, x_N\}$; (2) true class labels $T = \{t_1, t_2, \dots, t_N\}$; (3) Lagrange multiplier α ; (4) support threshold $minSupp$; and (5) growth ratio threshold $minRatio$. In particular, $minSupp$ and $minRatio$ are parameters for contrast pattern mining. The first step is to extract contrast patterns from the training data. Then the explanation units S for all the instances are constructed based on the contrast patterns extracted, such that every contrast pattern p represents the meaning of an explanation unit, and it is 1 if an instance x matches the corresponding contrast pattern. After the explanation units for all training instances are ready, X, T and S are used as inputs to train ExplNet by minimizing Eq. (8.2). The contrast patterns are only generated at training time and they are not needed for testing.

We adopt a random forest based pattern mining method Shang et al., 2016 to generate contrast patterns due to its efficiency. For multi-class scenario, the contrast patterns for each class is generated in a One-vs-the-Rest manner. There also exist other techniques for contrast pattern mining, such as GcGrowth Li et al., 2005, subgroup discovery Atzmueller, 2015 and exceptional model mining Duivesteijn, Feelders, and Knobbe, 2016. We refer readers to Dong and Bailey, 2012 for further discussion about contrast patterns.

Algorithm 11: Learning process of ExplNet with contrast pattern explanation units

Input: $X = \{x_1, x_2, \dots, x_N\}, T = \{t_1, t_2, \dots, t_N\}, \alpha, minSupp, minRatio$
Output: ExplNet, represented by the weight matrix

- 1 $DP = miningPatterns_DPclass(X, T);$
- 2 Select contrast patterns CP by filtering DP using $minSupp, minRatio$;
- 3 $K = |CP|;$
- 4 $S = \{ \};$
- 5 **for** $x_i \in X$ **do**
- 6 $s_i = (s_{i,1}, s_{i,2}, \dots, s_{i,K});$
- 7 **for** $j = 0, 1, \dots, K$ **do**
- 8 **if** x_i matches $p_j \in CP$ **then**
- 9 $s_{i,j} = 1$
- 10 **end**
- 11 **else**
- 12 $s_{i,j} = 0$
- 13 **end**
- 14 **end**
- 15 add s_i to S
- 16 **end**
- 17 For any instance x_i , use s_i as the true explanation units for x_i , together with label t_i . Train ExplNet by solving Eq (8.2) with $E = \{E_1, E_2, \dots, E_N\}, X, T$ and α ;
- 18 return ExplNet;

8.3.5 Motivation example

We introduce an example to show the effectiveness of ExplNet. Consider the Titanic dataset¹ of 29 features, where the goal is to predict whether a passenger would survive

¹<http://web.stanford.edu/class/archive/cs/cs109/cs109.1166/problem12.html>

or not. Assume that one possible explanation for a prediction is that: “if a passenger is female and the passenger class is 3, then the passenger can survive”. We compare two neural networks: ExplNet and a classification-only neural network (NN) with the explanation subnetwork removed from ExplNet. For instances of passenger information ‘female=1’ and ‘passengerClass=3’, ExplNet achieves 99% accuracy, much higher compared to the 91% of the classification-only NN. This implies that, with proper supervision of the expert explanation, the process of learning explanations can boost the accuracy of the final classifier.

8.4 Experiments

In this section, we evaluate and discuss the performance of ExplNet with several baselines in different scenarios.

8.4.1 Datasets

We conduct the experiments using UCI datasets and the Titanic datasets that are described in Table 8.3.

Dataset	#inst	#nomAttr	#numAttr	#classes
Adult	48842	8	6	2
Balloon	16	4	0	2
Blood	758	0	4	2
Breast-cancer	596	8	0	2
Diabetes	768	0	8	2
Magic	19020	0	11	2
Musk	6598	0	168	2
Titanic	3772	22	7	2
Vote	435	16	0	2

#inst - number of instances, #nomAttr - number of nominal attributes, #numAttr - number of numeric attributes, #classes - number of classes

TABLE 8.3: UCI datasets description.

8.4.2 Baseline methods

We evaluate the following methods: **ExplNet** - our proposed network where the true explanations for training are from experts or domain knowledge; **ExplNet(CP)** - our proposed network where contrast patterns are used as true explanation for training; **NN** - Standard neural network without explanation subnetwork; **NB** - Naive Bayes; **AdaBoost** - AdaBoost; **SVM** - Support vector machine with the RBF kernel; **RF** - Random forest.

In order to evaluate our approach and the baselines, we need to create a ground truth for the explanations. To do this we manipulate the datasets by building an interpretable oracle classifier from the data and relabeling the instances with the classifier, so that the ground truth explanation is available from the interpretable oracle classifier and can be used to compute a reliability score.

In order to train ExplNet, we need expert explanations, and this can be accomplished by either using the ground truth from the oracle classifier (what **ExplNet** does), or by mining contrast patterns using the training data (what **ExplNet(CP)** does).

Considering that traditional methods are not able to work with expert explanations, we also evaluate the baselines in a way such that the expert explanations can be used.

The explanation units and class labels are combined together forming a multi-label classification task where the explanation units are treated as the second type of class label. We wrap the baseline methods with a One-vs-the-Rest (OvR) classifier such that each class/explanation unit is represented by one and one classifier only. This is the most commonly used strategy for multi-label classification. The baselines with this procedure are denoted by the “*” symbol (e.g., **RF*** denotes a random forest model trained with expert explanations in a multi-label setting).

8.4.3 Experimental settings

Parameter optimization via a nested validation process during the training process is a common approach for classification tasks, however, it is difficult for explanation systems. This is mostly due to the fact that true model explanations are always unavailable when deploying such systems in real-world scenarios. A solution to select the parameters is to generate a parameter setting using validation datasets whose true model explanations are available, and then apply the same setting for any new dataset. To simulate such a process, we apply ExplNet with different parameter settings of three major parameters $minSupp$, $minRatio$ and α on four separate UCI datasets (*annealing*, *crx*, *iris* and *ILPD*, which are different from those used for evaluation) as validation datasets, then the recommended parameter settings ($minSupp = 0.15$, $minRatio = 3$, $\alpha = 1$) that achieve the best accuracy score on average over the validation datasets is used for all the datasets whose results are reported for comparison and analysis in this section. 10-fold validation is used to evaluate ExplNet and other baseline methods, and the averaged metric scores are reported.

8.4.4 Metrics

We evaluate two metrics: accuracy and reliability. Accuracy is the percentage of the number of correct predictions over the number of all the predictions in the test dataset. Given the true/expert explanation units ($e_1^*, e_2^*, \dots, e_K^*$) and the pattern representations (see Section 3) $P^* = (p_1^*, p_2^*, \dots, p_K^*)$, the reliability of the prediction of x with explanation units ($e_1, e_2, \dots, e_O$) and the pattern representations $P = (p_1, p_2, \dots, p_O)$ is defined as the Jaccard similarity of the triggered pattern sets. Note that if the true/expert explanations are available, O is equal to K and P is identical to P^* ; if the true/expert explanations are not available to the model and contrast patterns are used (see Section 8.3.4), P is different from P^* as P is obtained via pattern mining and P^* is pre-defined for evaluation purposes. Formally, the reliability is defined as:

$$Reliability(x) = \frac{|mds(Q^*) \cap mds(Q)|}{|mds(Q^*) \cup mds(Q)|} \quad (8.6)$$

where $Q^* \subset P^*$ is the pattern set such that $p_i^* \in P^*$ and $e_i^* = 1$, and Q is defined similarly using P . $mds(Q)$ is the union of all matching instances for every $q \in Q$.

For our proposed method, explanations are obtained directly from the output of the explanation subnetwork. For baseline methods, explanations are generated by LIME Ribeiro, Singh, and Guestrin, 2016c (with decision trees as the local model) where the models and their predictions are inputs for LIME and the pattern explanations are obtained from the local decision trees.

8.4.5 Evaluation with expert explanation from a single tree

In this part, we evaluate the proposed method and the baselines in a scenario where explanations are extracted from expert/domain knowledge. We simulate the experts by an

oracle decision tree classifier, which is a common procedure for testing explanation systems Ribeiro, Singh, and Guestrin, 2016c. For each dataset, an oracle decision tree is built on the training data, then it is applied to relabel all instances, and the explanations are obtained from the prediction paths of the decision tree. An explanation unit e is triggered ($e = 1$) when an instance is classified using the corresponding path of the model. The oracle explanation units for training data are given to the model for learning, while the oracle explanation units for the test data are used to compare the explanation outputs from ExplNet and evaluate the reliability. To simulate the scenario when the expert explanations are not available, ExplNet with contrast patterns is used.

In this case, there exist K explanation units $e_1, e_2, \dots, e_K$, where each explanation unit corresponds to a prediction path of the oracle decision tree that has K paths in total. Among these explanation units, only one explanation can be triggered, since the paths in a decision tree are exclusive.

The reliability results are reported in Table 8.5, while the accuracy results can be found in Table 8.6. Overall, ExplNet outperforms other baselines in terms of reliability across all tested datasets, and it also achieves comparable accuracy results to the baselines (the average accuracy is 0.05 % less than the best accuracy). In the situations when the expert explanations are not available, we mine contrast patterns as the explanation units and it still outperforms other baselines in terms of reliability, while its accuracy is just 2.1% less than the best accuracy of the baselines.

8.4.6 Evaluation with expert patterns from committee of trees

In this part, we simulate the scenario where expert explanations come from a bag of trees where the bagging technique is used. A set of trees (a bagging classifier with trees) are built on each training dataset, then all instances for each dataset are re-labeled by the classifier, and the explanation units $e_1, e_2, \dots, e_K$ are obtained from these trees. Assume there are m trees and each tree contains K_m prediction paths, then $K = \sum_i^m K_m$ and the explanation units are the union of the prediction paths of each tree.

Except for the generation process of the expert explanation units, we follow the same settings and procedures as in the previous example.

The results are reported in Tables 8.7 and 8.8. It can be observed that ExplNet again outperforms the baselines across all datasets in terms of reliability, and it is able to win in 5 out of 9 datasets and achieves the best accuracy on average. Though the metrics with contrast pattern explanations are lower than the metrics with expert explanations, it again shows that contrast patterns could be a good choice when expert explanations are not available, as it still outperforms the baselines in terms of reliability, and achieves comparable accuracy with the baselines. It can be concluded that ExplNet works well, especially when the underlying classification logical explanations are complex.

8.4.7 What happens when ExplNet and other models disagree?

It would be interesting to see how the explanation reliability changes when ExplNet and baseline methods make different predictions. Though it is expected that models make correct prediction when the explanations are reliable and vice versa, most models fail to achieve it. In this part, we analyze the scenario when ExplNet disagrees with the random forest (RF), and similar conclusions can be made for other baseline methods. For each dataset D , the test instances are divided into two groups D_1 and D_2 , where D_1 contains instances on which ExplNet and RF make the same predictions, while D_2 contains instances on which ExplNet and RF make different predictions. We investigate the reliability and accuracy on D_1 and D_2 for ExplNet and RF. When ExplNet and RF

Inputs	ExplNet	ExplNet(CP)	NN	NB	AdaBoost	SVM	RF	NN*	NB*	AdaBoost*	SVM*	RF*
instances?	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
labels?	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
expert explanations?	Y	N	N	N	N	N	N	Y	Y	Y	Y	Y

TABLE 8.4: Input required for the proposed method and the baselines. “Y” represents “Required” and “N” represents “Not required”. ExplNet - our proposed network where the explanation units are from the oracle; ExplNet(CP) - our proposed network where the explanation units are from contrast patterns; NN - Neural network without explanation subnetwork; NB - Naive Bayes; AdaBoost - AdaBoost; SVM - Support vector machine with RBF kernel; RF - Random forest. “*” denotes models trained with expert explanations in a multi-label setting.

Dataset	Reliability											
	ExplNet	ExplNet(CP)	NN	NB	AdaBoost	SVM	RF	NN*	NB*	AdaBoost*	SVM*	RF*
adult	0.92	0.82	0.33	0.81	0.63	0.4	0.69	0.69	0.67	0.68	0.37	0.56
blood	0.87	0.69	0.54	0.58	0.25	0.63	0.21	0.48	0.49	0.52	0.22	0.59
breast-cancer	1.0	0.88	0.22	0.68	0.59	0.56	0.67	0.54	0.36	0.65	0.32	0.74
diabetes	0.65	0.54	0.48	0.52	0.36	0.49	0.37	0.23	0.29	0.44	0.27	0.37
ionosphere	0.81	0.75	0.35	0.61	0.31	0.64	0.69	0.57	0.38	0.61	0.25	0.67
magic	0.81	0.74	0.27	0.65	0.48	0.4	0.2	0.56	0.42	0.56	0.64	0.51
musk	0.94	0.86	0.21	0.72	0.34	0.27	0.3	0.47	0.47	0.59	0.49	0.4
titanic	0.94	0.88	0.48	0.79	0.41	0.5	0.52	0.78	0.6	0.76	0.45	0.57
vote	1.0	0.94	0.6	0.78	0.44	0.84	0.68	0.78	0.46	0.73	0.31	0.82
average	0.88	0.79	0.39	0.68	0.68	0.42	0.53	0.57	0.46	0.62	0.37	0.58

TABLE 8.5: Reliability results where expert explanations are extracted from decision trees (with the best values in **bold**). ExplNet - our proposed network where the explanation units are from the oracle; ExplNet(CP) - our proposed network where the explanation units are from contrast patterns; NN - Neural network without explanation subnetwork; NB - Naive Bayes; AdaBoost - AdaBoost; SVM - Support vector machine with RBF kernel; RF - Random forest. "*" denotes models trained with expert explanations in a multi-label setting.

agree on predictions from D_1 , ExplNet can achieve 95% accuracy on average compared to 96% for RF, and the reliability is 0.95 and 0.43 for ExplNet and RF respectively. When ExplNet and RF disagree on group D_2 , ExplNet can achieve 12% accuracy on average compared with that of 13% for RF, and the reliability is 0.12 and 0.28 for ExplNet and RF respectively. The results suggest that ExplNet is more likely to make correct predictions when the explanations are also reliable and vice versa. While for other baselines such as random forest, the correlation is less strong.

8.4.8 Impact of α

In this part, we investigate the impact of α and analyze the relationship of accuracy and reliability in the oracle explanation setting (similar trends can be found using contrast pattern explanations).

We vary the value of α in $[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^0, 10^1, 10^2, 10^3, 10^4]$. The results for individual datasets are shown in Figure 8.2. The overall trend is that the performance of both accuracy and reliability increases and then decreases when increasing α . These results confirm the benefit of our joint learning process: joint learning of explanations and class labels tends to yield better results than individual learning tasks (a small value of α approximates the results of the individual learning class label task, while a large value approximates the results of learning explanations only). One possible reason is that the hard sharing structure greatly reduces the risk of overfitting, thus when the two tasks are jointly learned, the proposed model ExplNet is able to find a representation that reduces the chance of overfitting to the original tasks.

8.5 Conclusion

In this chapter, we studied the problem of simultaneously addressing prediction performance and interpretability of classification models, and proposed a neural network based multi-task learning framework, namely ExplNet, to jointly learn classification and explanations. ExplNet has two subnetworks on top of a shared base network, with one classification subnetwork and one explanation network. The two subnetworks were trained jointly with a Lagrange function of a classification loss and an explanation loss. We also propose to use contrast patterns extracted from the training data as alternative explanations for scenarios where ground truth explanations are not obtainable. We empirically show that ExplNet can learn both accurate and reliable predictions. ExplNet can be easily applied to applications where explanations are as important as accuracy for decision making.

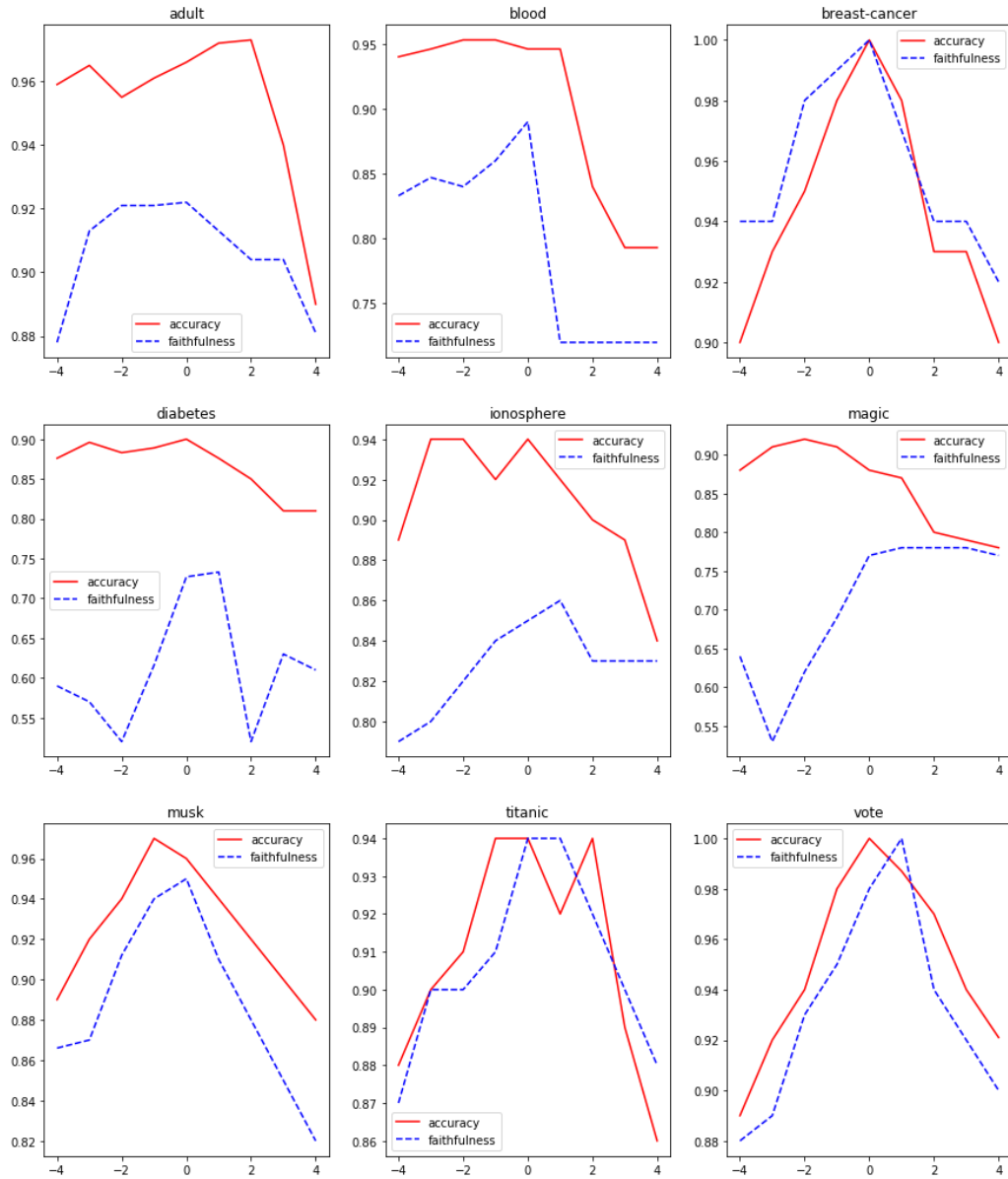


FIGURE 8.2: Impact of α . The accuracy and reliability results with respect to varying α in $[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^0, 10^1, 10^2, 10^3, 10^4]$

Accuracy(%)												
Dataset	ExplNet	ExplNet(CP)	NN	NB	AdaBoost	SVM	RF	NN*	NB*	AdaBoost*	SVM*	RF*
adult	94.7	94.2	94.7	94.7	56.5	56.5	95.6	95.6	87.8	87.8	82.6	82.6
blood	94.6	93.2	86.6	86.6	86.6	86.6	100.0	100.0	90.0	90.0	99.3	99.3
breast-cancer	100.0	98.1	98.2	98.2	100.0	100.0	100.0	100.0	93.1	93.1	93.1	91.3
diabetes	90.2	88.4	86.3	86.3	83.1	83.1	85.7	85.7	88.3	88.3	92.8	83.1
ionosphere	91.5	90.5	92.9	92.9	91.5	91.5	97.1	97.1	92.9	92.9	91.5	94.3
magic	90.7	86.7	90.8	90.8	77.9	77.9	93.2	93.2	89.6	89.6	84.5	83.9
musk	97.0	95.6	94.9	94.9	76.5	76.5	94.5	94.5	88.7	88.7	84.9	82.5
titanic	94.4	93.1	87.0	87.0	82.5	82.5	91.3	91.3	91.3	91.3	88.5	88.2
vote	100.0	98.5	100.0	100.0	95.4	95.4	100.0	100.0	81.6	81.6	89.6	91.9
average	94.7	93.1	92.3	92.3	83.3	83.3	95.2	95.2	89.2	89.2	89.6	88.5

TABLE 8.6: Accuracy results with expert explanations are extracted from decision trees (with the best values in **bold**). ExplNet - our proposed network where the explanation units are from the oracle; ExplNet(CP) - our proposed network where the explanation units are from contrast patterns; NN - Neural network without explanation subnetwork; NB - Naive Bayes; AdaBoost - AdaBoost; SVM - Support vector machine with RBF kernel; RF - Random forest. “*” denotes models trained with expert explanations in a multi-label setting.

Dataset	Reliability											
	ExpINet	ExpINet(CP)	NN	NB	AdaBoost	SVM	RF	NN*	NB*	AdaBoost*	SVM*	RF*
adult	0.78	0.71	0.33	0.65	0.63	0.15	0.69	0.54	0.66	0.64	0.68	0.44
blood	0.83	0.73	0.29	0.07	0.6	0.4	0.4	0.51	0.24	0.4	0.65	0.55
breast-cancer	0.72	0.65	0.25	0.57	0.22	0.65	0.6	0.4	0.53	0.12	0.56	0.01
diabetes	0.81	0.75	0.67	0.0	0.52	0.01	0.56	0.58	0.25	0.01	0.31	0.03
ionosphere	0.8	0.78	0.52	0.36	0.38	0.01	0.25	0.76	0.68	0.4	0.33	0.4
magic	0.91	0.81	0.23	0.23	0.42	0.11	0.46	0.81	0.25	0.32	0.37	0.26
musk	0.89	0.82	0.26	0.42	0.32	0.01	0.62	0.54	0.3	0.41	0.52	0.58
titanic	0.78	0.71	0.51	0.23	0.41	0.24	0.54	0.49	0.24	0.29	0.54	0.24
vote	0.91	0.84	0.58	0.24	0.29	0.13	0.54	0.59	0.42	0.38	0.29	0.21
average	0.8	0.75	0.4	0.3	0.4	0.1	0.5	0.5	0.3	0.3	0.4	0.3

TABLE 8.7: Reliability results where expert explanations come from decision trees (with the best values in **bold**). ExpINet - our proposed network where the explanation units are from the oracle; ExpINet(CP) - our proposed network where the explanation units are from contrast patterns; NN - Neural network without explanation subnetwork; NB - Naive Bayes; AdaBoost - AdaBoost; SVM - Support vector machine with RBF kernel; RF - Random forest. “*” denotes models trained with expert explanations in a multi-label setting.

Accuracy(%)												
Dataset	ExplNet	ExplNet(CP)	NN	NB	AdaBoost	SVM	RF	NN*	NB*	AdaBoost*	SVM*	RF*
adult	97.3	95.6	98.2	98.2	53.9	53.9	99.1	99.1	90.4	90.4	83.4	83.4
blood	90.6	88.9	84.0	84.0	83.3	83.3	94.0	95.0	84.0	84.0	94.3	95.0
breast-cancer	98.2	95.5	96.5	96.5	91.3	91.3	96.5	96.5	86.2	86.2	87.9	87.9
diabetes	94.1	91.3	90.9	90.9	88.9	88.9	90.2	90.2	93.5	93.5	86.3	86.3
ionosphere	95.7	92.1	97.1	97.1	90.1	90.1	94.3	94.3	88.7	88.7	87.3	92.9
magic	95.5	91.4	93.1	93.1	79.5	79.5	95.3	95.3	91.8	91.8	84.8	85.0
musk	97.6	92.6	96.7	96.7	80.5	80.5	94.8	94.8	88.7	94.8	92.8	92.4
titanic	94.5	91.4	91.2	90.2	94.2	92.4	93.8	92.4	95.9	93.8	95.8	91.2
vote	94.8	90.8	94.5	95.4	97.4	97.4	92.5	95.1	98.5	94.3	92.4	95.4
average	95.3	92.2	93.5	93.5	84.3	84.1	94.5	94.7	90.8	90.8	89.4	89.9

TABLE 8.8: Accuracy results where expert explanations come from decision trees (with the best values in **bold**). ExplNet - our proposed network where the explanation units are from the oracle; ExplNet(CP) - our proposed network where the explanation units are from contrast patterns; NN - Neural network without explanation subnetwork; NB - Naïve Bayes; AdaBoost - AdaBoost; SVM - Support vector machine with RBF kernel; RF - Random forest. “*” denotes models trained with expert explanations in a multi-label setting.

Chapter 9

Conclusion and Future work

9.1 Conclusion

Explainability of machine learning models can facilitate the successful deployments or help develop trust of the models in the domains like where the interpretability is desired. Many complex models like neural networks, random forests are able to achieve the state-of-the-art accuracy. However, their deployments can be limited due to their nature of being black boxes. In many cases, machine learning models are used to explain data, or to generate valuable knowledge. For example, in the business area, the machine learning models are expected to generate meaningful business knowledge from data, and a business manager is more likely to accept the recommendations if they are explained in business terms (Bose and Mahapatra, 2001). This chapter discusses the use of explanations and their applications in different area.

The main goals of explanations in machine learning models includes: (1) establishing trust of the predictions of the model, and (2) enabling models to cooperate with expert explanations to make reliable predictions.

The first goal requires the machine learning models to be able to provide explanations and the second goal requires them to be able to utilize the expert explanations. The first goal can be achieved by (1) using interpretable models such as decision tree and linear models, and (2) using model-agnostic methods as interpretable interface to provide explanations for black-box models.

We achieve the first goal by:

- We propose a pattern based method PALEX (see Chapter 5) to generate explanations for black-box models. Given a model and a query instance being explained, PALEX first generates a synthetic neighborhood around the query using random perturbations, then labels the synthetic instances using the model, finally PALEX mines contrast patterns from the synthetic neighborhood and selects top K patterns as the final explanations. The experiments show that PALEX is able to achieve high faithfulness and support multi-explanation scenarios.
- We investigate the crucial process of generating synthetic neighbors for local explanations methods, as different synthetic neighbors can result in explanations of different quality, and in many cases, the random perturbation does work well. We analyze the relationship of local intrinsic dimensionality (LID) and the quality of explanations, and propose a LID based method LEAP (see Chapter 6) to generate the synthetic neighbors such that the generated synthetic neighbors are more effective than the ones generated by other baselines.
- For the explanations to be meaningful, the features should also be interpretable. Some feature transformation methods (e.g., the kernel methods) increase the complexity of the feature, which makes the new features difficult to explain. We also

present a framework to do interpretable feature engineering using patterns. The framework consists of three main steps: finding patterns, selecting representative patterns and transforming original features into new features using pattern based mapping functions. The new pattern based features are discriminative and interpretable.

The second goal of explanations can be achieved by: (1) using expert explanations as additional features such that the explanations are used as input features for learning, but the expert explanations are not required for predicting, (2) using expert explanations as additional class labels, then the learning problem is transferred to multi-class learning where there are more than one class label, and (3) using expert explanations to perform feature engineering or data cleaning.

we achieve the second goal by:

- We propose a pattern based partition-wise linear models method xPWL (see Chapter 7) that can be trained together with expert explanations. xPWL divide the data space into several partitions, and each partition is represented by a pattern and is associated with a local linear model. The overall prediction for a query is a linear combination of the local linear models in the activated partitions. The model is interpretable and is able to work with expert explanations as a loss function in terms of explanations is part of the ultimate loss function. The results show that xPWL is able to make superior reliable predictions and achieve competitive accuracy comparing with the baseline methods.
- We then propose a neural network structure (see Chapter 8) that is able to jointly train the class labels and the explanations where the explanations can be treated as another label information. We fit a neural network in the framework of multi-task learning. The neural network starts with a set of shared layers and then split into two separate layers where one is for class label and the other is for explanations. The experiments suggest that the proposed method is able to make reliable predictions.

9.2 Future works

9.2.1 Collecting expert explanations

We have shown that a model that can be trained with the expert explanations is able to make reliable predictions such that the explanations for the predictions of a model match the expert knowledge or domain information. However, currently the expert explanations are still not available in practice. In this part, we discuss how to collect the expert explanations in several practical areas where the interpretability is desired.

For example, recommendation systems have been widely used in marketing. Companies like online bookstores, music or video online streaming providers employ the recommendation systems to analyze the tastes of customers and generate product/item recommendations to the customers. These systems can pull from reams of nuanced data in order to reach final results from the behaviors, actions, etc.. As mentioned in (Zhang and Iyengar, 2002), for recommendation systems, interpretability is a desired characteristic to be considered in addition to the accuracy, and it can enable marketing analysts to review and possibly modify the model. Nearest neighbor based methods are used to find the recommendations based on a customer's recent action in (Breese, Heckerman, and Kadie, 1998). Generalized linear models like logistic regression are also widely used to generate the recommendations as they are interpretable Cheng et al., 2016.

Expert explanations can be obtained via customer feedback or customer survey where the users are asked to provide information about what features are important for them. For example, when successfully recommending a video to a user, the user can be asked what features of the recommendation are considered.

The customers can also be provided with several explanation candidates and select the most accurate one. These candidates can be generated with explanation methods that support multi-explanation scenario such as PALEX.

In medical scenarios, the expert explanations in healthcare can be found from medical documents such as a doctor's report or any other form of diagnosis report. Then text mining techniques can be used to extract the expert explanations in required forms.

9.2.2 Using explanations to improve predictive accuracy

The explanations for the predictions of a model can reveal that logical flaw of the model if the explanation does not match the domain knowledge. An example is shown in (Ribeiro, Singh, and Guestrin, 2016c) where a model is trained to predict whether a given image is "husky" or "wolf", and the explanation for a prediction is "because the background is snow", which ignores the animal in the image.

A natural question is how to correct the logical flaw of machine learning models and consequently improve the models' accuracy. Due to the limited availability of explanations for the models, few research has been conducted in this direction. However, with the field of explainable machine learning continuing to mature, this could be a promising direction to improve the accuracy of machine learning models using the explanations, and eventually lead to successful deployment of the models in real-life applications.

Bibliography

- Aamodt, Agnar and Enric Plaza (1994). “Case-based reasoning: Foundational issues, methodological variations, and system approaches”. In: *AI Communications* 7.1, pp. 39–59.
- Adler, Philip, Casey Falk, Sorelle A Friedler, Gabriel Rybeck, Carlos Scheidegger, Brandon Smith, and Suresh Venkatasubramanian (2016). “Auditing black-box models for indirect influence”. In: *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, pp. 1–10.
- Adler, Philip, Casey Falk, Sorelle A Friedler, Tionney Nix, Gabriel Rybeck, Carlos Scheidegger, Brandon Smith, and Suresh Venkatasubramanian (2018). “Auditing black-box models for indirect influence”. In: *Knowledge and Information Systems* 54.1, pp. 95–122.
- Aggarwal, Charu C and Jiawei Han (2014). *Frequent pattern mining*. Springer.
- Agrawal, Rakesh and John C Shafer (1996). “Parallel mining of association rules”. In: *IEEE Transactions on knowledge and Data Engineering* 8.6, pp. 962–969.
- Agrawal, Rakesh and Ramakrishnan Srikant (1995). “Mining sequential patterns”. In: *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*. IEEE, pp. 3–14.
- Agrawal, Rakesh, Ramakrishnan Srikant, et al. (1994). “Fast algorithms for mining association rules”. In: *Proceedings of the 20th International Conference on Very Large Data Bases*. Vol. 1215, pp. 487–499.
- Altman, Naomi S (1992). “An introduction to kernel and nearest-neighbor nonparametric regression”. In: *The American Statistician* 46.3, pp. 175–185.
- Amsaleg, Laurent, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E. Houle, Ken-ichi Kawarabayashi, and Michael Nett (2018). “Extreme-value-theoretic estimation of local intrinsic dimensionality”. In: *Data Mining and Knowledge Discovery* 32.6, pp. 1768–1805.
- Amsaleg, Laurent, Oussama Chelly, Michael E. Houle, Ken-ichi Kawarabayashi, Miloš Radovanović, and Weeris Treeratanajaru (2019). “Intrinsic Dimensionality Estimation within Tight Localities”. In: *Proceedings of the 19th SIAM International Conference on Data Mining (SDM)*. SIAM, pp. 181–189.
- Arias-Castro, Ery, Guangliang Chen, Gilad Lerman, et al. (2011). “Spectral clustering based on local linear approximations”. In: *Electronic Journal of Statistics* 5, pp. 1537–1587.
- Atzmueller, Martin (2015). “Subgroup discovery”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5.1, pp. 35–49.
- Ayres, Jay, Jason Flannick, Johannes Gehrke, and Tomi Yiu (2002). “Sequential pattern mining using a bitmap representation”. In: *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 429–435.
- Baehrens, David, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller (2010). “How to explain individual classification decisions”. In: *Journal of Machine Learning Research* 11.Jun, pp. 1803–1831.

- Bailey, James and Elsa Loekito (2010). "Efficient incremental mining of contrast patterns in changing data". In: *Information Processing Letters* 110.3, pp. 88–92.
- Barakat, Nahla and Joachim Diederich (2005). "Eclectic rule-extraction from support vector machines". In: *International Journal of Computational Intelligence* 2.1, pp. 59–62.
- Bay, Stephen D and Michael J Pazzani (2001). "Detecting group differences: Mining contrast sets". In: *Data Mining and Knowledge Discovery* 5.3, pp. 213–246.
- Bayardo Jr, Roberto J (1998). "Efficiently mining long patterns from databases". In: *ACM Sigmod Record* 27.2, pp. 85–93.
- Beyer, Kevin, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft (1999). "When is "nearest neighbor" meaningful?" In: *International Conference on Database Theory*. Springer, pp. 217–235.
- Bose, Indranil and Radha K Mahapatra (2001). "Business data mining—a machine learning perspective". In: *Information & management* 39.3, pp. 211–225.
- Breese, John S, David Heckerman, and Carl Kadie (1998). "Empirical analysis of predictive algorithms for collaborative filtering". In: *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., pp. 43–52.
- Breiman, Leo (2001). "Random forests". In: *Machine Learning* 45.1, pp. 5–32.
- (2017). *Classification and Regression Trees*. Routledge.
- Brown, Tom B, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer (2017). "Adversarial patch". In: *arXiv preprint arXiv:1712.09665*.
- Caruana, Rich, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad (2015). "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission". In: *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 1721–1730.
- Cheng, Heng-Tze, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. (2016). "Wide & deep learning for recommender systems". In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, pp. 7–10.
- Cheng, Hong, Xifeng Yan, Jiawei Han, and Chih-Wei Hsu (2007). "Discriminative frequent pattern analysis for effective classification". In: *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, pp. 716–725.
- Cheng, Hong, Xifeng Yan, Jiawei Han, and S Yu Philip (2008). "Direct discriminative pattern mining for effective classification". In: *2008 IEEE 24th International Conference on Data Engineering*. IEEE, pp. 169–178.
- Chow, C and Cong Liu (1968). "Approximating discrete probability distributions with dependence trees". In: *IEEE Transactions on Information Theory* 14.3, pp. 462–467.
- Collobert, Ronan and Jason Weston (2008). "A unified architecture for natural language processing: Deep neural networks with multitask learning". In: *Proceedings of the 25th International Conference on Machine learning*. ACM, pp. 160–167.
- Cong, Gao, Kian-Lee Tan, Anthony KH Tung, and Xin Xu (2005). "Mining top-k covering rule groups for gene expression data". In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. ACM, pp. 670–681.
- Craven, Mark W and Jude W Shavlik (1996). "Extracting tree-structured representations of trained networks". In: *Advances in Neural Information Processing Systems*, pp. 24–30.
- Deng, Li, Geoffrey Hinton, and Brian Kingsbury (2013). "New types of deep neural network learning for speech recognition and related applications: An overview". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, pp. 8599–8603.

- Deshpande, Mukund, Michihiro Kuramochi, Nikil Wale, and George Karypis (2005). "Frequent substructure-based approaches for classifying chemical compounds". In: *IEEE Transactions on Knowledge and Data Engineering* 17.8, pp. 1036–1050.
- Dong, Guozhu and James Bailey (2012). *Contrast data mining: concepts, algorithms, and applications*. CRC Press.
- Dong, Guozhu and Jinyan Li (1999). "Efficient mining of emerging patterns: Discovering trends and differences". In: *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 43–52.
- Dong, Guozhu and Vahid Taslimitehrani (2015). "Pattern-aided regression modeling and prediction model analysis". In: *IEEE Transactions on Knowledge and Data Engineering* 27.9, pp. 2452–2465.
- (2016). "Pattern aided classification". In: *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, pp. 225–233.
- Dong, Guozhu, Xiuzhen Zhang, Limsoon Wong, and Jinyan Li (1999). "CAEP: Classification by aggregating emerging patterns". In: *International Conference on Discovery Science*. Springer, pp. 30–42.
- Dougherty, James, Ron Kohavi, Mehran Sahami, et al. (1995). "Supervised and unsupervised discretization of continuous features". In: *Proceedings of the 12th International Conference on Machine Learning*. Vol. 12, pp. 194–202.
- Duivesteijn, Wouter, Ad J Feelders, and Arno Knobbe (2016). "Exceptional model mining". In: *Data Mining and Knowledge Discovery* 30.1, pp. 47–98.
- Duivesteijn, Wouter and Julia Thaele (2014). "Understanding where your classifier does (not) work—the SCAPE model class for EMM". In: *Proceedings of the 14th IEEE International Conference on Data Mining*. IEEE, pp. 809–814.
- Ezghari, Soufiane, Rachid Benouini, Azeddine Zahi, and Khalid Zenkouar (2017). "Learning efficient and interpretable prototypes from data for nearest neighbor classification method". In: *2017 Intelligent Systems and Computer Vision*. IEEE, pp. 1–7.
- Fan, Hongjian and Kotagiri Ramamohanarao (2006). "Fast discovery and the generalization of strong jumping emerging patterns for building compact and accurate classifiers". In: *IEEE Transactions on Knowledge and Data Engineering* 18.6, pp. 721–737.
- Fei, Hongliang and Jun Huan (2008). "Structure feature selection for graph classification". In: *Proceedings of the 17th ACM Conference on Information and Knowledge Management*. ACM, pp. 991–1000.
- Fisher, Aaron, Cynthia Rudin, and Francesca Dominici (2018). "All models are wrong but many are useful: Variable importance for black-box, proprietary, or misspecified prediction models, using model class reliance". In: *arXiv preprint arXiv:1801.01489*.
- Fong, Ruth C and Andrea Vedaldi (2017). "Interpretable Explanations of Black Boxes by Meaningful Perturbation". In: *Proceedings of International Conference on Computer Vision (ICCV)*. IEEE, pp. 3449–3457.
- Freitas, Alex A (2014). "Comprehensible classification models: a position paper". In: *ACM SIGKDD Explorations Newsletter* 15.1, pp. 1–10.
- Friedman, Jerome H (2001). "Greedy function approximation: a gradient boosting machine". In: *Annals of Statistics*, pp. 1189–1232.
- Fuchs, Karen, Jan Gertheiss, and Gerhard Tutz (2015). "Nearest neighbor ensembles for functional data with interpretable feature selection". In: *Chemometrics and Intelligent Laboratory Systems* 146, pp. 186–197.
- Fung, Glenn, Sathyakama Sandilya, and R Bharat Rao (2005). "Rule extraction from linear support vector machines". In: *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM, pp. 32–40.

- Gay, Dominique and Marc Boullé (2012). “A Bayesian approach for classification rule mining in quantitative databases”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 243–259.
- Goldstein, Alex, Adam Kapelner, Justin Bleich, and Emil Pitkin (2015). “Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation”. In: *Journal of Computational and Graphical Statistics* 24.1, pp. 44–65.
- Goodfellow, Ian J, Jonathon Shlens, and Christian Szegedy (2014). “Explaining and harnessing adversarial examples”. In: *CoRR*.
- Goodman, Bryce and Seth Flaxman (2016). “EU regulations on algorithmic decision-making and a “right to explanation””. In: *ICML Workshop on Human Interpretability in Machine Learning (WHI 2016)*.
- Gregorutti, Baptiste, Bertrand Michel, and Philippe Saint-Pierre (2017). “Correlation and variable importance in random forests”. In: *Statistics and Computing* 27.3, pp. 659–678.
- Gretton, Arthur, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola (2007). “A kernel method for the two-sample-problem”. In: *Advances in Neural Information Processing Systems*, pp. 513–520.
- Guidotti, Riccardo, Anna Monreale, Salvatore Ruggieri, Dino Pedreschi, Franco Turini, and Fosca Giannotti (2018). “Local rule-based explanations of black box decision systems”. In: *arXiv preprint arXiv:1805.10820*.
- Guns, Tias, Siegfried Nijssen, and Luc De Raedt (2013). “k-Pattern set mining under constraints”. In: *IEEE Transactions on Knowledge and Data Engineering* 25.2, pp. 402–418.
- Guralnik, V and G Karypis (2001). “A scalable algorithm for clustering sequential data”. In: *Proceedings 2001 IEEE International Conference on Data Mining*. IEEE, pp. 179–186.
- Han, Jiawei, Jian Pei, and Micheline Kamber (2011). *Data mining: concepts and techniques*. Elsevier.
- Han, Jiawei, Jian Pei, and Yiwen Yin (2000). “Mining frequent patterns without candidate generation”. In: *ACM Sigmod Record*. Vol. 29. 2. ACM, pp. 1–12.
- Han, Jiawei, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu (2000). “FreeSpan: frequent pattern-projected sequential pattern mining”. In: *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 355–359.
- Han, Jiawei, Jian Pei, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and MC Hsu (2001). “Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth”. In: *proceedings of the 17th International Conference on Data Engineering*, pp. 215–224.
- Han, Jiawei, Hong Cheng, Dong Xin, and Xifeng Yan (2007). “Frequent pattern mining: current status and future directions”. In: *Data Mining and Knowledge Discovery* 15.1, pp. 55–86.
- Hastie, Trevor, Robert Tibshirani, Jerome Friedman, and James Franklin (2005). “The elements of statistical learning: data mining, inference and prediction”. In: *The Mathematical Intelligencer* 27.2, pp. 83–85.
- Henelius, Andreas, Kai Puolamäki, Henrik Boström, Lars Asker, and Panagiotis Papapetrou (2014). “A peek into the black box: exploring classifiers by randomization”. In: *Data Mining and Knowledge Discovery* 28.5-6, pp. 1503–1529.
- Henelius, Andreas, Kai Puolamäki, Isak Karlsson, Jing Zhao, Lars Asker, Henrik Boström, and Panagiotis Papapetrou (2015). “Goldeneye++: A closer look into the black box”. In: *International Symposium on Statistical Learning and Data Sciences*. Springer, pp. 96–105.

- Houle, Michael E. (2017a). “Local intrinsic dimensionality I: an extreme-value-theoretic foundation for similarity applications”. In: *Proceedings of the 10th International Conference on Similarity Search and Applications*. Springer, pp. 64–79.
- (2017b). “Local intrinsic dimensionality II: multivariate analysis and distributional support”. In: *Proceedings of the 10th International Conference on Similarity Search and Applications*. Springer, pp. 80–95.
- Huan, Jun, Wei Wang, and Jan Prins (2003). “Efficient mining of frequent subgraphs in the presence of isomorphism”. In: *Third IEEE International Conference on Data Mining*. IEEE, pp. 549–552.
- Huan, Jun, Wei Wang, Deepak Bandyopadhyay, Jack Snoeyink, Jan Prins, and Alexander Tropsha (2004). “Mining protein family specific residue packing patterns from protein structure graphs”. In: *Proceedings of the 8th Annual International Conference on Research in Computational Molecular Biology*. ACM, pp. 308–315.
- Inokuchi, Akihiro, Takashi Washio, and Hiroshi Motoda (2000). “An apriori-based algorithm for mining frequent substructures from graph data”. In: *Principles of Data Mining and Knowledge Discovery*, pp. 13–23.
- Ji, Xiaonan, James Bailey, and Guozhu Dong (2007). “Mining minimal distinguishing subsequence patterns with gap constraints”. In: *Knowledge and Information Systems* 11.3, pp. 259–286.
- Jose, Cijo, Prasoon Goyal, Parv Aggrwal, and Manik Varma (2013). “Local deep kernel learning for efficient non-linear svm prediction”. In: *Proceedings of the 30th International Conference on Machine Learning*, pp. 486–494.
- Jovanovic, Milos, Sandro Radovanovic, Milan Vukicevic, Sven Van Poucke, and Boris Delibasic (2016). “Building interpretable predictive models for pediatric hospital readmission using Tree-Lasso logistic regression”. In: *Artificial Intelligence in Medicine* 72, pp. 12–21.
- Kang, Sori and Kotagiri Ramamohanarao (2014). “A Robust Classifier for Imbalanced Datasets”. In: *Advances in Knowledge Discovery and Data Mining*. Springer International Publishing, pp. 212–223.
- Karmarkar, Narendra (1984). “A new polynomial-time algorithm for linear programming”. In: *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*. ACM, pp. 302–311.
- Kaufman, Shachar, Saharon Rosset, Claudia Perlich, and Ori Stitelman (2012). “Leakage in data mining: Formulation, detection, and avoidance”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6.4, p. 15.
- Kim, Been, Rajiv Khanna, and Oluwasanmi O Koyejo (2016). “Examples are not enough, learn to criticize! criticism for interpretability”. In: *Advances in Neural Information Processing Systems*, pp. 2280–2288.
- Kim, Been, Cynthia Rudin, and Julie A Shah (2014). “The bayesian case model: A generative approach for case-based reasoning and prototype classification”. In: *Advances in Neural Information Processing Systems*, pp. 1952–1960.
- Kingma, Diederik P and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*.
- Kocbek, Primož, Nino Fijacko, Cristina Soguero-Ruiz, Karl Øyvind Mikalsen, Uros Maver, Petra Povalej Brzan, Andraz Stozer, Robert Jenssen, Stein Olav Skrøvseth, and Gregor Stiglic (2019). “Maximizing Interpretability and Cost-Effectiveness of Surgical Site Infection (SSI) Predictive Models Using Feature-Specific Regularized Logistic Regression on Preoperative Temporal Data”. In: *Computational and Mathematical Methods in Medicine* 2019.

- Koh, Pang Wei and Percy Liang (2017). “Understanding black-box predictions via influence functions”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 1885–1894.
- Kohavi, Ron (1995). “The power of decision tables”. In: *Proceedings of the 8th European Conference on Machine Learning*, pp. 174–189.
- Kroupa, Pavel, Caroline Morton, Kerlann Le Calvez, and Matt Williams (2019). “Assessing K-Nearest Neighbours algorithm for simple, interpretable time-to-event survival predictions over a range of simulated datasets”. In: *2019 IEEE 32nd International Symposium on Computer-Based Medical Systems (CBMS)*. IEEE, pp. 367–372.
- Kuramochi, Michihiro and George Karypis (2001). “Frequent subgraph discovery”. In: *Proceedings 2001 IEEE International Conference on Data Mining*. IEEE, pp. 313–320.
- Kurfess, Franz J. (2000). “Neural Networks and Structured Knowledge: Rule Extraction and Applications”. In: *Applied Intelligence* 12.1, pp. 7–13. ISSN: 1573-7497. DOI: [10.1023/A:1008344602888](https://doi.org/10.1023/A:1008344602888). URL: <http://dx.doi.org/10.1023/A:1008344602888>.
- Ladicky, Lubor and Philip Torr (2011). “Locally linear support vector machines”. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 985–992.
- Lakkaraju, Himabindu, Stephen H Bach, and Jure Leskovec (2016). “Interpretable decision sets: A joint framework for description and prediction”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 1675–1684.
- Laugel, Thibault, Marie-Jeanne Lesot, Christophe Marsala, Xavier Renard, and Marcin Detyniecki (2017). “Inverse Classification for Comparison-based Interpretability in Machine Learning”. In: *arXiv preprint arXiv:1712.08443*.
- Laugel, Thibault, Xavier Renard, Marie-Jeanne Lesot, Christophe Marsala, and Marcin Detyniecki (2018). “Defining Locality for Surrogates in Post-hoc Interpretability”. In: *Workshop on Human Interpretability for Machine Learning (WHI)-International Conference on Machine Learning, 2018*.
- Letham, Benjamin, Cynthia Rudin, Tyler H McCormick, David Madigan, et al. (2015). “Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model”. In: *The Annals of Applied Statistics* 9.3, pp. 1350–1371.
- Li, Haiquan, Jinyan Li, Limsoon Wong, Mengling Feng, and Yap-Peng Tan (2005). “Relative risk and odds ratio: A data mining perspective”. In: *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, pp. 368–377.
- Li, Jinyan, Guozhu Dong, and Kotagiri Ramamohanarao (2001). “Making use of the most expressive jumping emerging patterns for classification”. In: *Knowledge and Information systems* 3.2, pp. 131–145.
- Li, Jinyan, Thomas Manoukian, Guozhu Dong, and Kotagiri Ramamohanarao (2004). “Incremental maintenance on the border of the space of emerging patterns”. In: *Data Mining and Knowledge Discovery* 9.1, pp. 89–116.
- Li, Wenmin, Jiawei Han, and Jian Pei (2001). “CMAR: Accurate and efficient classification based on multiple class-association rules”. In: *2001 Proceedings IEEE International Conference on Data Mining*. IEEE, pp. 369–376.
- Loekito, Elsa and James Bailey (2009). “Using highly expressive contrast patterns for classification-is it worthwhile?” In: *Advances in Knowledge Discovery and Data Mining*, pp. 483–490.

- Lv, Cui and Di-Rong Chen (2018). “Interpretable Functional Logistic Regression”. In: *Proceedings of the 2nd International Conference on Computer Science and Application Engineering*. ACM, p. 82.
- Ma, Bing Liu Wynne Hsu Yiming and Bing Liu (1998). “Integrating classification and association rule mining”. In: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*.
- Ma, Xingjun, Bo Li, Yisen Wang, Sarah M. Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Michael E. Houle, Dawn Song, and James Bailey (2018). “Characterizing Adversarial Subspaces Using Local Intrinsic Dimensionality”. In: *Proceedings of the 6th International Conference on Learning Representations (ICLR)*.
- Maron, Melvin Earl (1961). “Automatic indexing: an experimental inquiry”. In: *Journal of the ACM (JACM)* 8.3, pp. 404–417.
- Martens, David and Foster Provost (2011). “Explaining documents’ classifications”. In: *Center for Digital Economy Research*.
- Martens, David, Bart Baesens, Tony Van Gestel, and Jan Vanthienen (2007). “Comprehensible credit scoring models using rule extraction from support vector machines”. In: *European Journal of Operational Research* 183.3, pp. 1466–1476.
- Molnar, Christoph (2019). *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. <https://christophm.github.io/interpretable-ml-book/>.
- Oiwa, Hidekazu and Ryohei Fujimaki (2014). “Partition-wise linear models”. In: *Advances in Neural Information Processing Systems*, pp. 3527–3535.
- Papernot, Nicolas, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami (2017). “Practical black-box attacks against machine learning”. In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, pp. 506–519.
- Parikh, Neal, Stephen Boyd, et al. (2014). “Proximal algorithms”. In: *Foundations and Trends in Optimization* 1.3, pp. 127–239.
- Park, Jong Soo, Ming-Syan Chen, and Philip S Yu (1995). *An effective hash-based algorithm for mining association rules*, pp. 175–186.
- Pasquier, Nicolas, Yves Bastide, Rafik Taouil, and Lotfi Lakhal (1999). “Discovering frequent closed itemsets for association rules”. In: *International Conference on Database Theory*. Springer, pp. 398–416.
- Pastor, Eliana and Elena Baralis (2019). “Explaining black box models by means of local rules”. In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp. 510–517.
- Ramamohanarao, Kotagiri, James Bailey, and Hongjian Fan (2005). “Efficient mining of contrast patterns and their applications to classification”. In: *2005 3rd International Conference on Intelligent Sensing and Information Processing*. IEEE, pp. 39–47.
- Ramsundar, Bharath, Steven Kearnes, Patrick Riley, Dale Webster, David Konerding, and Vijay Pande (2015). “Massively multitask networks for drug discovery”. In: *arXiv preprint arXiv:1502.02072*.
- Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2016a). “Model-Agnostic Interpretability of Machine Learning”. In: *ICML Workshop on Human Interpretability in Machine Learning (WHI 2016)*.
- (2016b). “Nothing Else Matters: Model-Agnostic Explanations By Identifying Prediction Invariance”. In: *NIPS Workshop on Interpretable Machine Learning in Complex Systems*.
- (2016c). “Why Should I Trust You?: Explaining the Predictions of Any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 1135–1144.

- Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2018). “Anchors: High-precision model-agnostic explanations”. In: *AAAI Conference on Artificial Intelligence*.
- Robnik-Šikonja, Marko and Igor Kononenko (2008). “Explaining classifications for individual instances”. In: *IEEE Transactions on Knowledge and Data Engineering* 20.5, pp. 589–600.
- Romano, Simone, Oussama Chelly, Xuan Vinh Nguyen, James Bailey, and Michael E. Houle (2016). “Measuring Dependency via Intrinsic Dimensionality”. In: *Proc. 23rd International Conference on Pattern Recognition (ICPR)*, pp. 1207–1212.
- Ruder, Sebastian (2016). “An overview of gradient descent optimization algorithms”. In: *CoRR*. arXiv: 1609.04747. URL: <http://arxiv.org/abs/1609.04747>.
- (2017). “An overview of multi-task learning in deep neural networks”. In: *arXiv preprint arXiv:1706.05098*.
- Salvador, Amaia, Xavier Giró-i Nieto, Ferran Marqués, and Shin’ichi Satoh (2016). “Faster r-cnn features for instance search”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 9–16.
- Selvaraju, Ramprasaath R, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra (2017). “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization.” In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 618–626.
- Shang, Jingbo, Wenzhu Tong, Jian Peng, and Jiawei Han (2016). “Dpclass: An effective but concise discriminative patterns-based classification framework”. In: *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, pp. 567–575.
- Slade, Stephen (1991). “Case-based reasoning: A research paradigm”. In: *AI Magazine* 12.1, pp. 42–42.
- Smets, Koen and Jilles Vreeken (2012). “Slim: Directly mining descriptive patterns”. In: *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, pp. 236–247.
- Srikant, Ramakrishnan and Rakesh Agrawal (1996). “Mining sequential patterns: Generalizations and performance improvements”. In: pp. 1–17.
- Storn, Rainer and Kenneth Price (1997). “Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces”. In: *Journal of Global Optimization* 11.4, pp. 341–359.
- Štrumbelj, Erik and Igor Kononenko (2014). “Explaining prediction models and individual predictions with feature contributions”. In: *Knowledge and Information Systems* 41.3, pp. 647–665.
- Su, Jiawei, Danilo Vasconcellos Vargas, and Kouichi Sakurai (2019). “One pixel attack for fooling deep neural networks”. In: *IEEE Transactions on Evolutionary Computation*.
- Szegedy, Christian, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus (2013). “Intriguing properties of neural networks”. In: *arXiv preprint arXiv:1312.6199*.
- Ting, Roger Ming Hieng and James Bailey (2006). “Mining minimal contrast subgraph patterns”. In: *Proceedings of the 2006 SIAM International Conference on Data Mining*. SIAM, pp. 639–643.
- Toivonen, Hannu et al. (1996). “Sampling large databases for association rules”. In: *Proceedings of the 22th International Conference on Very Large Data Bases*. Vol. 96, pp. 134–145.
- Uysal, Alper Kürşat (2018). “Feature Selection for Comment Spam Filtering on YouTube”. In: *Data Science and Applications* 1.1, pp. 4–8.
- Vanetik, Natalia, Ehud Gudes, and Solomon Eyal Shimony (2002). “Computing frequent graph patterns from semistructured data”. In: *2002 IEEE International Conference on Data Mining*. IEEE, pp. 458–465.

- Vreeken, Jilles, Matthijs Van Leeuwen, and Arno Siebes (2011). “Krimp: mining itemsets that compress”. In: *Data Mining and Knowledge Discovery* 23.1, pp. 169–214.
- Wachter, Sandra, Brent Mittelstadt, and Chris Russell (2017). “Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR”. In: *Harvard Journal of Law & Technology* 31, p. 841.
- Wang, Chao and Srinivasan Parthasarathy (2006). “Summarizing itemset patterns using probabilistic models”. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 730–735.
- Wang, Fulton and Cynthia Rudin (2015). “Falling rule lists”. In: *Artificial Intelligence and Statistics*, pp. 1013–1022.
- Wang, Jianyong and George Karypis (2005). “HARMONY: Efficiently mining the best rules for classification”. In: *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, pp. 205–216.
- Wang, Jianyong, Yuzhou Zhang, Lizhu Zhou, George Karypis, and Charu C Aggarwal (2007). “Discriminating subsequence discovery for sequence clustering”. In: *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, pp. 605–610.
- Wang, Joseph and Venkatesh Saligrama (2012). “Local supervised learning through space partitioning”. In: *Advances in Neural Information Processing Systems*, pp. 91–99.
- Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas (2016). “Dueling Network Architectures for Deep Reinforcement Learning”. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML’16. New York, NY, USA: JMLR.org, pp. 1995–2003. URL: <http://dl.acm.org/citation.cfm?id=3045390.3045601>.
- Wolpert, David H (1992). “Stacked generalization”. In: *Neural Networks* 5.2, pp. 241–259.
- Xin, Dong, Hong Cheng, Xifeng Yan, and Jiawei Han (2006). “Extracting redundancy-aware top-k patterns”. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 444–453.
- Xing, Eric P, Michael I Jordan, Stuart J Russell, and Andrew Y Ng (2003). “Distance metric learning with application to clustering with side-information”. In: *Advances in Neural Information Processing Systems*, pp. 521–528.
- Yan, Xifeng and Jiawei Han (2002). “gspan: Graph-based substructure pattern mining”. In: *2002 IEEE International Conference on Data Mining*. IEEE, pp. 721–724.
- Yan, Xifeng, Hong Cheng, Jiawei Han, and Dong Xin (2005). “Summarizing itemset patterns: a profile-based approach”. In: *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM, pp. 314–323.
- Yin, Xiaoxin and Jiawei Han (2003). “CPAR: Classification based on predictive association rules”. In: *Proceedings of the 2003 SIAM International Conference on Data Mining*. SIAM, pp. 331–335.
- Zaïane, Osmar R, Yang Wang, Randy Goebel, and Gregory Taylor (2006). “Frequent subsequence-based protein localization”. In: *International Workshop on Data Mining for Biomedical Applications*. Springer, pp. 35–47.
- Zaki, Mohammed J (2001). “SPADE: An efficient algorithm for mining frequent sequences”. In: *Machine Learning* 42.1, pp. 31–60.
- Zaki, Mohammed Javeed (2000). “Scalable algorithms for association mining”. In: *IEEE Transactions on Knowledge and Data Engineering* 12.3, pp. 372–390.
- Zhang, Tong and Vijay S Iyengar (2002). “Recommender systems using linear classifiers”. In: *Journal of Machine Learning Research* 2.Feb, pp. 313–334.

- Zhang, Xiuzhen, Guozhu Dong, and Kotagiri Ramamohanarao (2001). “Building behaviour knowledge space to make classification decision”. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, pp. 488–494.
- Zhou, Cheng, Boris Cule, and Bart Goethals (2016). “Pattern based sequence classification”. In: *IEEE Transactions on Knowledge and Data Engineering* 28.5, pp. 1285–1298.