Searching For Relevant Features To Classify Crew Pairing Problems

Challenges of Applying Machine Learning Methods

Master's thesis in Applied Data Science

Jin GUO

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

Searching For Relevant Features To Classify Crew Pairing Problems

Challenges of Applying Machine Learning Methods

Jin GUO

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Searching For Relevant Features To Classify Crew Pairing Problems
Challenges of Applying Machine Learning Methods
Jin GUO

Supervisor: Dag Wedelin, Department of Computer Science and Engineering
Advisor: Björn Thalén, Jeppesen, a Boeing Company
Examiner: Richard Johansson, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

Abstract

Machine learning (ML) is an emerging technology. Jeppesen, a leader of commercial optimization products in the airline industry, has started exploring ML methods to facilitate optimization algorithm development. This thesis investigates one of the company's products, the crew pairing optimizer. The optimizer can use different algorithms to solve crew pairing problems, and the thesis looks into what features of a pairing problem influence algorithm selection, i.e. the best choice of algorithm for a problem, based on the performance of different algorithms. With little prior knowledge about features of pairing problems and their relation with algorithm performance, using ML, the thesis first generates over twenty features, and then uses different feature selection methods to find the most informative feature subsets. Each feature subset is then fed into multiple classifiers to test its robustness. Besides ML, the thesis also includes statistical analysis as a comparison. The thesis has some interesting findings, including a subset of features that might influence algorithm performance. However, none of the methods used can find a feature subset to accurately classify the pairing problems by the best performing algorithm. The thesis discusses possible reasons for the results. It also lists what to consider before applying ML to real-world problems.

Keywords: Machine learning, airline crew pairing, feature selection, algorithm selection, classification.

Acknowledgements

I would first like to thank my thesis supervisor, Prof. Dag Wedelin of the Department of Computer Science and Engineering at Chalmers University of Technology. Dag is the most brilliant teacher I have ever met, and his instructions have always inspired me to enter the next level.

I am thankful to the experts at Jeppesen. My advisor Björn Thalén collected the data, shared his domain knowledge with me, offered me great help at the feature extraction stage, and answered countless questions from me. Special thanks to Erik Berglund for his valuable suggestions on the project. I would also like to thank Fredrik Altenstedt, Mattias Grönkvist and Joakim Karlsson for their kind help.

I must express my gratitude to all the teachers and classmates on my master program, who have continuously helped and inspired me along the way. I am forever thankful to our program director, PhD Richard Johansson. Without his encouragement and help, I would not have embarked upon a career in Data Science.

Finally, I would like to thank my family and friends. Special thanks to my best friend of ten years, PhD Tianyu, Jiang, for being a ray of sunshine during my undergraduate and graduate studies. My deep gratitude to my partner, Fan Gao, for his faith in me. His support has given me courage to overcome any difficulty.

Jin Guo, Gothenburg, October 2019

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Scope of the project . . . 1
1.2 Challenges . . . 2

2 Background and Related Work **3**
2.1 The Pairing Problem . . . 3
2.2 Performance Based Algorithm Selection . . . 3

3 Theory: Classification and Feature Selection **5**
3.1 Classification Modeling . . . 5
3.1.1 Logistic regression . . . 6
3.1.2 Classification tree . . . 9
3.1.3 K nearest neighbors . . . 11
3.1.4 Tree ensembles . . . 13
3.2 Feature Selection . . . 14
3.2.1 Filter . . . 14
3.2.2 Wrapper . . . 15
3.2.3 Embedded . . . 16
3.3 Low-Dimensional Representation of Data . . . 17
3.4 Evaluation of Selected Feature Subsets . . . 17

4 Data: Collection and Exploration **19**
4.1 Input Features . . . 19
4.1.1 Features extracted from graphs . . . 20
4.1.2 Static features from meta data . . . 22
4.2 The Response . . . 23
4.2.1 The classes: three algorithms . . . 23
4.2.2 Labeling method . . . 23
4.3 Data Collection Process . . . 25
4.4 Data Exploration . . . 26

5 Methods: Machine Learning and Statistical Analysis **31**
5.1 Statistical Analysis . . . 35

6 Results **37**
6.1 Machine Learning Analysis 37
6.1.1 Feature selection 37
6.1.2 Performance of classifiers 40
6.1.3 The best feature subset 41
6.2 Statistical Analysis 43

7 Conclusion and Discussion **49**
7.1 Result Analysis 49
7.2 Findings 50
7.3 Future work 51

Bibliography **53**

A Appendix 1 **I**

List of Figures

3.1 Binary logistic regression 7
3.2 A possible classification tree for the toy data of ice cream sales 10

4.1 The DAG representation of a connection table example 20
4.2 Pairwise correlations of data collected from OAG 26
4.3 Histogram of features 27
4.4 Boxplot of features 28
4.5 Scatter plots of the first two and three PCA components, computed from all features of all observations 28

5.1 Visualization of the split and use of data 34

6.1 CV accuracies for all methods with varying number of features to be selected 38
6.2 Confusion matrices of CV predictions 39
6.3 Visualization of the number of times the features are selected by the ten folds 39
6.4 Confusion matrices of the test set predictions 41
6.5 Feature importance ranked by the RF model 42
6.6 Scatter plots of the first two and three PCA components, computed from selected features of all observations 43
6.7 CV accuracies for Logistic Lasso regression with varying α 44
6.8 Visualization of the number of times the features are selected by the ten folds using LLR 45
6.9 Lars path of Logistic Lasso Regression 47

A.1 Pair plots of features I
A.2 Accumulative variance explained by the PCA components using all observations and all features II
A.3 Accumulative variance explained by the PCA components using all observations and selected subset of features II
A.4 Visualization of the single decision tree model III

List of Tables

3.1 Toy data of daily ice cream sales of a supermarket 7
3.2 Toy data of ice cream sales 10

4.1 Class distribution 27

5.1 Analysis pipeline using machine learning methods 31
5.2 Feature selection methods used in ML analysis 32

6.1 Model accuracies on the test set 40
6.2 Comparing feature importance indicated by RF feature importance and CV feature selection 42
6.3 Numbers of features selected by Logistic Lasso Regression with varying *alpha* 44
6.4 Sum of the numbers of times a feature is selected with varying *alpha* 45
6.5 Logistic Lasso Regression Results 46

1

Introduction

The crew pairing optimizer is one of Jeppesen's main products. It is an advanced and complicated system aiming to solve the pairing problem: optimizing schedules of flight duties for crew members. The optimizer can use different optimization algorithms to solve a pairing problem. Depending on the features of a pairing problem, such as characteristics of the considered fleet, work-time rules of crew members etc., the performance of different optimization algorithms might differ, influencing what is the best choice of algorithm, i.e. which algorithm should be selected for the problem. Knowledge about what features of a pairing problem impact on algorithm selection can therefore help with understandings of the pairing problem, as well as the future development of the pairing optimizer. However, the procedures of the pairing optimizer are computationally heavy. Due to the complexity of the optimizer, we cannot easily observe what features of a pairing problem influence the performance of different algorithms. On top of this, the relation between problem features and algorithm performance is usually quite complex. As an initial trial to model such a relation, this thesis project attempts to find the relevant features.

Three different algorithms are considered in this project. Therefore, the thesis attempts to extract features and relates them to the three algorithms. The goal of this thesis is thus to find the features that can predict which of the three algorithms will perform the best for a pairing problem. In other words, the project searches for relevant features to classify crew pairing problems by the best performing algorithm.

The project mainly uses Machine Learning (ML) to find such features. ML is an emerging technology, and people from the field of mathematical optimization, including experts at Jeppesen, are experimenting with ML methods to facilitate the development of optimization algorithms.

1.1 Scope of the project

This project collects data of pairing problems, extracts candidate features from the data, labels the problems by the best performing algorithms, visualizes the data from different aspects, selects the best subsets of features with different feature se-

lection methods, builds different models using the selected feature subsets to classify the data, and finally evaluates model performance. This thesis does not aim to build a system to suggest which algorithm is the best to use for a pairing problem. As a first step toward that direction, and more importantly, for the purpose of improved understanding, the thesis purely attempts to find the relevant features that can classify the pairing problems.

1.2 Challenges

The first challenge is to collect a sufficient data set. The pairing problems are complex and large, and it will take a significant amount of time to run multiple optimization algorithms over hundreds of cases. Besides execution time of the algorithms, data accessibility also restricts the scope of our data set. Historical flight schedules are not freely accessible.

The second challenge is that the definition of 'best performing algorithm' is open to discussion. The evaluation of algorithm performance involves both execution time and the quality of the algorithm's final solution. Moreover, the performance difference of different algorithms on a case may be negligible.

The third challenge is to find the relevant features, as well as where to find these features. The pairing problem is a complex problem, and the pairing optimizer is a complicated system. Algorithm performance might be influenced by subtle details of a pairing problem and vary in unpredictable ways. Therefore, there is no guarantee that the considered features can differentiate the cases.

The last and fundamental challenge is that it is not certain whether there exists a relation between any measurable features of pairing problems and algorithm performance differences. Combined with the above challenges, it is an open question whether we can achieve our goal by approaching it as a classification task.

Although the project has the above mentioned challenges, it is still worth an attempt. If we manage to find some relevant features that relate to algorithm performance difference, it will provide the company with some potential directions for future optimization algorithm development. Even if we cannot find any significantly relevant feature with our approach, we test the feasibility of the idea. As Jeppesen is at an early stage of integrating ML into algorithm development, the collected data and findings from the data can serve as a start point for future experiments.

2

Background and Related Work

2.1 The Pairing Problem

Each airline company has a set of flights to be operated during a time period [6], and has one or several bases where its crews are located. The term pairing refers to a sequence of flights, arranged 'for an unspecified crew member starting and ending at the same crew base' [1]. The pairing problem is the problem of finding the optimal set of pairings, so that every flight is covered by a group of crew members required by the assigned fleet of the flight, and that all legal and labor union rules are followed, while minimizing the total costs. These factors are mathematically modeled into a cost function, and the optimizer works to minimize the total cost of the cost function. Each individual flight in a pairing is called a leg. Ideally, crew members should be working on all the legs. When a leg transports a crew member as a passenger, it is called a 'deadhead'. We hope to avoid deadheads, but sometimes they are necessary.

The crew pairing problem is a hard optimization problem. To name some of the difficulties, firstly, pairing problems often have very complex constraints, covering both legal rules and collective bargaining agreements. Secondly, solving the pairing problem usually means finding the best set of pairings from a large number of possible pairings, i.e. a large solution space that is heavily constrained.

2.2 Performance Based Algorithm Selection

For optimization algorithms, performance testing is 'notoriously difficult' [28]. There is still a strong lack of understanding of the relation between instance characteristics and algorithm performance, as well as a lack of objective measurement of algorithm performance [34].

The common approach for algorithm performance testing has been to test and compare different algorithms on a few well-studied test instances in the benchmark libraries. Benchmark cases for performance testing are collected in a convenient way,

which often do not well represent the set of real problems solved by the algorithm [15]. As a result, algorithms are tuned to perform well on these few test cases, and the quality of algorithm assessment purely depends on how diverse and representative the test cases are. Researchers and readers are left with limited understanding of how the algorithms will perform outside the benchmark libraries [28].

In contrast with the likely biased performance analysis of the common approach, researchers have proposed different approaches to select the best performing algorithm for an optimization problem. Hooker (1995) suggests matching problem characteristics, i.e. problem features, to algorithm performance by controlled experiments with the characteristics. The method generates problem instances by tuning only one characteristic while fixing all others each time. For instance, one may generate a series of instances of sizes $n_1, n_2, \ldots, n_m$ while fixing all other problem characteristics, and then repeat the process with another characteristic. After generating data in this controlled way, one compares algorithm performance on the instances. Similarly, Gomes and Selman (1997) also generated instances in a controlled manner by varying the problem characteristics, including inherent problem structure and computational difficulty. However, these methods require researchers to know what features to control with prior to data generation, which is not feasible for problems where one only has limited knowledge.

Rice (1976) proposed a different procedure for algorithm selection, which has four main components. They include collecting a set of instances of an optimization problem, a set of algorithms that can be used to solve the problem, measurable characteristics of the instances, and algorithm performance over the different instances. Following Rice's idea, Smith-Miles et al. (2014) developed a methodology to objectively assess and compare the performance of different optimization algorithms. The methodology involves creating a broad instance space, extracting features, predicting algorithm performance and analyzing algorithmic power. They approached the task as a classification problem, where features of an instance are matched to the best-performing algorithm on that case. Using this methodology, [29] successfully predicts the best performing algorithm for job shop scheduling problems, [27] for quadratic assignment, [30] for timetabling, and [33] [32] for traveling salesman.

Together with other authors, Rice acknowledged in his later work that 'the way problem features affect methods is complex and algorithm selection might depend in an unstable way on the features actually used' [25]. Moreover, very little literature focuses on the topic of how to devise a set of potentially interesting features [31]. Nevertheless, machine learning has helped with finding measurable features that can indicate algorithm performance for many optimization problems ([11, 12, 18, 27, 29, 30, 33, 32]). All of these projects analyze well-known academic optimization problems. This thesis intends to apply and adapt the methodology of Smith-Miles et al. (2014) to the real life optimization problem of crew pairing.

3

Theory: Classification and Feature Selection

This chapter covers theory about classification, feature selection, and evaluation of selected feature subsets. For classification, we discuss both the concept in general and the considered classifiers. For feature selection, we define and explain three kinds of feature selection methods used in the project. A short discussion of Principal Component Analysis (PCA) is also included, as the method is used to visualize our data. We do not include theory about pairing optimization algorithms, as the thesis does not look inside the pairing optimizer. Readers do not need details of crew pairing or the pairing optimization algorithms to understand the contents of this thesis.

3.1 Classification Modeling

The technical term for classifying observations using labeled inputs is called classification. Classification models are also called classifiers. Some possible examples of classification problems include using labeled car and non-car pictures to predict whether a new picture contains a car, or using historical payment records to predict whether a customer will pay back a debt.

Classification is a kind of supervised learning problem, as all inputs in the data have known outputs, and outputs are used to supervise the learning process of models. Supervised learning learns to model the relation between the input and the output. Inputs are also called features, independent variables or predictors interchangeably, while outputs are often called responses or the dependent variable.

Besides classification, the other kind of supervised learning is regression. The difference between the two lies in the type of response. A regression problem has a quantitative, usually a continuous variable as response. For example, if we predict how many boxes of ice cream will be sold in a supermarket tomorrow using historical sales data, or to predict tomorrow's stock price using historical stock market data, it is a regression problem. In contrast, a classification problem has a categorical

response. In other words, the possible values of the response are discrete and can be completely enumerated.

The type of a classifier determines the kind of input and output relation that can be learned. As we do not have prior knowledge about our data, it is hard to choose a proper classifier directly. Therefore, we implement multiple classifiers. The remaining part of section 3.1 discusses classification models of Logistic Regression, classification tree, K Nearest Neighbors (KNN), and tree ensembles including Random Forest (RF), Extra Random Trees (ERT) and Boosted Trees. Logistic regression and classification tree are discussed in more detail. Logistic regression is a basic classification model. Classification tree is a popular model as it is intuitive. It is also the basis for tree ensembles.

3.1.1 Logistic regression

We use the two-class (binary) scenario to explain logistic regression. Binary classification means there are only two possible categories in the responses. The multinomial logistic regression for multiple-class logistic regression is a straightforward extension of the binary one. See [16] for information about multinomial logistic regression.

The logistic regression model adapts the linear regression model to classification problems. Therefore, as a preparation to understand logistic regression, we explain linear regression first. The linear regression model is one of the most basic and classic model for supervised learning problems. In a linear regression model, you assign weights $\boldsymbol{w}$ to the features of inputs $\boldsymbol{X}$, and predict the responses $\boldsymbol{y}$ using the sums of the weighted features:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} \tag{3.1}$$

Normally, we also include a constant term (intercept) in a linear regression model. We include a constant by adding a constant feature to $\boldsymbol{X}$, taking the value one for all observations $\boldsymbol{X_i}$ in $\boldsymbol{X}$. An example of a linear regression model can be built based on the toy data in Table 3.1 (ignoring the last row of Table 3.1 for now). We use temperature to predict how many boxes of ice cream are sold in a day in a supermarket. A constant is also included in the model. A possible relation between $\boldsymbol{X_i}$ and y_i can be $y_i = \boldsymbol{X_i}\boldsymbol{w} = 20 \times constant + 10 \times temperature = 20 + 10 \times temperature$. Therefore, if tomorrow's temperature forecast is 25 $°C$, our prediction for the number of boxes of ice cream that will be sold tomorrow in the supermarket is $20 + 10 \times 25 = 270$.

Sometimes we do not want an exact sales prediction, but rather the answer to whether we should worry about not having enough stock for today's sales. For instance, the supermarket has a normal daily stock level of 250 boxes of ice cream, and is interested in knowing how likely that the normal stock is not enough for the potential sales in a day, which can be measured by the probability of $(y > 250) = True$, i.e. by $p(y > 250)$. For such cases, logistic regression is a suitable choice.

observation: i	1	2	3	4	5	6
$Feature_0$: constant	1	1	1	1	1	1
$Feature_1$: temperature ($°C$)	0	10	20	23	30	40
y: boxes of ice cream	20	120	220	250	320	420
$y > 250$	False	False	False	False	True	True

Table 3.1: Toy data of daily ice cream sales of a supermarket

Similar to linear regression, logistic regression uses linear combinations of the input features to predict the responses. However, logistic regression does not directly return the predicted $\boldsymbol{X_i w}$. Instead, it maps $\boldsymbol{X_i w}$ to $p(\boldsymbol{X_i w} > threshold)$, i.e. the probability of $\boldsymbol{X_i w}$ larger than a threshold using a link function f:

$$p(\boldsymbol{X_i w} > threshold) = f(\boldsymbol{X_i w}) \tag{3.2}$$

With regard to our ice cream example, a logistic regression model maps $\boldsymbol{X_i w}$ to $p(\boldsymbol{X_i w} > threshold)$, and uses $p(\boldsymbol{X_i w} > threshold)$ as the estimation for $p(y_i > 250)$, the probability of not having enough stock of ice cream for day i. With increasing daily sales prediction, the probability of not having enough stock should also increase.

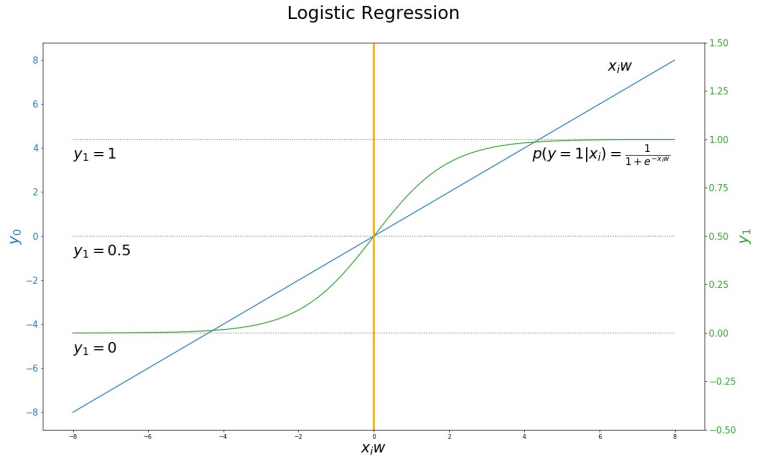

Figure 3.1: Binary logistic regression

So what kind of link function f should be chosen? Probabilities can only range from zero to one, while linear combination of input features can have a much wider range, as shown in our ice cream example. We need a one-to one mapping of all points on the straight line $\boldsymbol{X_i w}$ to a curve that lies within the (0,1) range, something like the mapping of the blue line to the green curve shown in Figure 3.1, where increasing $\boldsymbol{X_i w}$ corresponds to increasing $p(\boldsymbol{X_i w} > threshold)$. The x-axis in Figure 3.1 represents $\boldsymbol{X_i w}$. There are two y-axis. y_0 corresponds to the blue straight line, while y_1 corresponds to the green curve. The blue line shows the changes of $\boldsymbol{X_i w}$

against the x-axis. The green curve represents the change of $p(\boldsymbol{X_i w} > threshold)$ against the x-axis. The larger the $\boldsymbol{X_i w}$, the closer $p(\boldsymbol{X_i w} > threshold)$ is to one.

Now that we have the probabilities, how do we classify observations? How to answer the question of whether we should or should not worry about not having enough stock? For binary classification, logistic regression assigns the i^{th} observation to the class with higher probability. As the stock is either enough or not, we have two possible classes. Therefore, if $p(\boldsymbol{X_i w} > threshold)$ is higher than $1/2 = 0.5$, $p(\boldsymbol{X_i w} > threshold)$ will be larger than $p(\boldsymbol{X_i w} \leq threshold)$, meaning $p(y > 250) > p(y \leq 250)$. In this case we think the stock is not enough for the day. Otherwise our prediction is the stock is enough. Due to the nature of logistic regression's link function, the threshold of $\boldsymbol{X_i w} = 0$ corresponds to the $p = 0.5$ division point of an observation being either of the classes. If the 250 daily stock level is changed to 270, we can easily mitigate it by adjusting the constant term in $\boldsymbol{w}$ without changing the threshold of $\boldsymbol{X_i w} = 0$. As shown in Figure 3.1, divided by the orange line in the middle (the $\boldsymbol{X_i w} = 0$ line), all observations with $\boldsymbol{X_i w} > 0$ has a larger than 0.5 probability of going over the stock level, and can be classified as the 'not enough stock' class.

We now explain logistic regression in more technical terms. It uses the logit as its link function. The logit (log odds) of the response taking the value one is modeled by the following function:

$$\log\left(\frac{p(y_i = 1|\boldsymbol{X_i})}{1 - p(y_i = 1|\boldsymbol{X_i})}\right) = w_0 + w_1 x_{i1} + \cdots + w_p x_{ip} = \boldsymbol{X_i w} \tag{3.3}$$

where $\boldsymbol{X_i} = (1, x_1 \cdots, x_p)$ denotes the $p + 1$ features, including the constant, of the $i^t h$ input in the data. y_i denotes the i^{th} output. $\boldsymbol{w} = (w_0 \cdots, w_p)$ are the $p + 1$ feature weights. $p(y_i = 1|\boldsymbol{X_i})$ represents the probability of the i^{th} observation being a 'class one' object given $\boldsymbol{X_i}$, while $1 - p(y_i = 1|\boldsymbol{X_i})$ represents the probability of the i^{th} observation being a non-class-one object given $\boldsymbol{X_i}$. As we are interested in $p(y_i = 1|\boldsymbol{X_i})$, we transform equation (3.3) by using $\boldsymbol{X_i w}$ to represent $\mathrm{p}(y_i = 1|\boldsymbol{X_i})$, and we get function (3.4). Function (3.4) is equivalent to the sigmoid function:

$$\mathrm{p}(y_i = 1|\boldsymbol{X_i}) = \frac{\mathrm{e}^{w_0 + w_1 x_{i1} + \cdots + w_p x_{ip}}}{1 + \mathrm{e}^{w_0 + w_1 x_{i1} + \cdots + w_p x_{ip}}} = \frac{1}{1 + e^{-\boldsymbol{X_i w}}} \tag{3.4}$$

It is clear that the value range of function (3.4) is (0,1), which is the expected range of a probability. Using the $y_i \in \{-1, 1\}$ notation as class labels, and making use of the relation that $p(y = 1|\boldsymbol{X}) + p(y = -1|\boldsymbol{X}) = 1$, , we can see it always holds that:

$$p\left(y_i|\boldsymbol{X_i}\right) = \frac{1}{1 + e^{-y_i(\boldsymbol{X_i w})}} \tag{3.5}$$

Now we can use $\boldsymbol{X_i w}$ to calculate the probability of seeing y_i being either of the two classes given $\boldsymbol{X_i}$.

We refer to the ice cream sales example again to show you how logistic regression uses feature weights $\boldsymbol{w}$ to classify a case. The 'not enough stock of ice cream' is 'class one'. Looking at the data in Table 3.1, we can see that when $temperature > 23\ °C$, the boxes of ice cream sold in a day exceed the normal stock level of 250 boxes. When $temperature < 23\ °C$, the normal stock is enough to cover the daily sales. Therefore, $\boldsymbol{X_i w} = -230 + 10 \times temperature$ can help us with the classification. When $temperature > 23\ °C$, $\boldsymbol{X_i w} = -230 + 10 \times temperature > 0$, and $p\left(y_i = 1 | \boldsymbol{X_i}\right) > 0.5$. It means when the temperature is higher than 23 $°C$, the probability of not having enough stock is higher than 0.5, and the logistic regression model will predict the observation as a 'class one' object.

So how do we train a logistic regression model to get the weights? Assuming the independence of observations, the model formulates the maximum likelihood estimation by multiplying all the $p\left(y_i | \boldsymbol{X_i}\right)$'s. The set of $w_0, w_1, \ldots, w_p$ that maximizes the likelihood of observing such sequence of y_i's given $\boldsymbol{X_i}$'s is the set of parameters we are looking for. Finding the $\boldsymbol{w}$ that maximizes the likelihood is equivalent to finding the $\boldsymbol{w}$ that minimizes the negative of the likelihood. Therefore, the loss function of logistic regression, i.e. the negative log likelihood is formulated as:

$$\text{loss}(\boldsymbol{y}, \boldsymbol{w}, \boldsymbol{X}) = \sum_{i}^{N} \log\left(1 + e^{-y_i(\boldsymbol{X_i w})}\right) \tag{3.6}$$

We train an LR model by finding the set of weights $\boldsymbol{w}$ that minimizes the loss function (3.4).

3.1.2 Classification tree

The Classification Tree is a simple and easy to interpret model. To show you the basic idea of a tree model, we revisit the ice cream sales example. It is likely that the sales of ice cream is also influenced by whether the day is a holiday. On a holiday, the supermarket probably can sell as much ice cream as on a non-holiday at lower temperature. The updated toy data is shown in Table 3.2. Unlike logistic regression, we normally do not include a constant term in a tree model. A possible classification tree for this data is shown in Figure 3.2. The tree keeps asking relevant yes or no questions, so that it becomes more and more certain how we should label an object. By firstly asking the question of whether the day is a holiday, we can divide the whole data set into two subsets. The two subsets' division temperature of having enough or not enough stock of ice cream differs. If the day is a holiday, we ask another question of whether the temperature is higher than certain $20°C$. If the answer is Yes, the normal daily stock of 250 boxes is not enough for the potential sales of the day. Otherwise, our prediction is enough. For the other subset, where the day is not a holiday, we ask the follow-up question of whether the temperature is higher than 23 $°C$. Again, if the answer is yes, we think the stock is not enough, otherwise enough. As we can observe from this example, a classification tree works well when the features interact with each other.

observation: i	1	2	3	4	5	6	7
$Feature_0$: holiday	True	True	True	False	False	False	False
$Feature_1$: temperature ($°C$)	7	21	24	21	22	22.5	24
y: boxes of ice cream	120	260	290	230	240	245	260
$y > 250$	False	True	True	False	False	False	True

Table 3.2: Toy data of ice cream sales

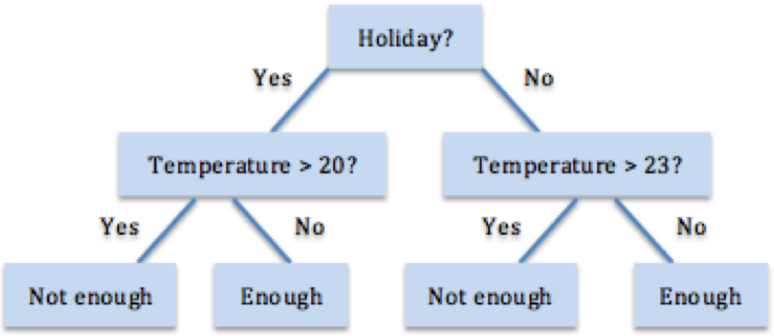

Figure 3.2: A possible classification tree for the toy data of ice cream sales

In general, a classification tree is built by asking a sequence of yes or no (binary) questions. More theoretically speaking, a tree segments the original data set into a number of small distinct non-overlapping subsets by recursively asking a series of binary questions. In the end, all observations are grouped into one of the subsets. Often, we have observations from more than one classes in the non-overlapping subsets. A tree classifies an observation by the majority class of the training observations in the subset the observation belongs, so every observation in the same subset gets the same prediction.

What questions should we ask? For a question like 'If $temperature > 20$', how do we come up with the number 20? It is not hard to see that ideally we want to end up with pure subsets, where each subset only contains one class. Therefore, we can rely on a method that measures purity of data to decide where to split the data. Gini index is one of such methods, and is defined as:

$$G = \sum_{k=1}^{K} \hat{p}_{mk} \left(1 - \hat{p}_{mk}\right) \tag{3.7}$$

where $\hat{p}_{mk}$ represents the proportion of class k training observations in subset m. The Gini index evaluates to zero when $\hat{p}_{mk}$ is zero or one. It means subset m either only contains class k object, or contains no class k object, indicating purity of the subset. We choose the feature and cutpoint that lead to the greatest possible reduction on Gini score.

Let us revisit the example in Table 3.2 to see how Gini index actually works.

Originally, the data has two classes, and there is only one set, so $m = 1$ and $k \in [-1, 1]$. The probability of an observation being class -1, i.e. sales not exceeding stock, is $\hat{p}_{1,-1} = \frac{4}{7}$. The probability of an observation being class 1, i.e. sales exceeding stock, is $\hat{p}_{1,1} = \frac{3}{7}$. So the Gini score of the original data set is $G_0 = \frac{4}{7} \times (1 - \frac{4}{7}) + \frac{3}{7} \times \left(1 - \frac{3}{7}\right) = \frac{24}{49}$. At the root, if we split by 'holiday', the holiday subset has 3 observations, and its Gini is $\frac{2}{3} \times (1 - \frac{2}{3}) + \frac{1}{3} \times \left(1 - \frac{1}{3}\right) = \frac{4}{9}$. The 'non-holiday' subset has four observations, and its Gini is $\frac{3}{4} \times (1 - \frac{3}{4}) + \frac{1}{4} \times \left(1 - \frac{1}{4}\right) = \frac{6}{16}$. The Gini score of the whole data set after the 'holiday' split, i.e. the weighted Gini score of the two subsets, is $G_1 = \frac{4}{9} \times \frac{3}{7} + \frac{6}{16} \times \frac{4}{7} = \frac{17}{42}$. In contrast, if we split by 'temperature $>$ 20' at the root, the 'hot' subset has six observations, and its Gini score is $\frac{3}{6} \times (1 - \frac{3}{6}) + \frac{3}{6} \times \left(1 - \frac{3}{6}\right) = \frac{1}{2}$. The 'not hot' subset has only one observation, so its Gini score is 0. The Gini score of the whole data set after the 'temperature>20' split, i.e. the weighted Gini score of the two subsets, is $G_1 = \frac{1}{2} \times \frac{6}{7} + 0 \times \frac{1}{7} = \frac{18}{42}$. $\frac{17}{42} < \frac{18}{42}$, so we achieve a larger reduction on the Gini of the whole data by splitting with 'holiday'. Therefore, 'holiday' is chosen as the first feature to split on by the tree. After the splits by temperature in both the 'holiday' and 'non-holiday' subsets, we end up with pure leaves. The final Gini score of the whole data is 0. Observe that the Gini decreases along the way.

From the explanations above, we can see that a classification tree can directly handle both binary and multiple-class problems, and take in both continuous and categorical features. It works well with interactions between features. However, a single tree is notoriously unstable: small changes in data can lead to a very different model [37]. In other words, a single tree tends to over-fit to the training data easily. If we keep asking questions, it is sometimes possible that we end up with a large tree with all final subsets containing only one observation. In this case we will have zero error rate on the training set, but its prediction power would be quite low, as the model is too adapted to the training set and cannot generate well to new data. One can prune this big tree, for example, to restrict the maximum depth of a tree, or the minimum number of samples in a subset. Although the pruned simple tree can lead to better prediction performance on future data than the big tree , it is at the cost of less precise modeling. In general, a single tree's predictive accuracy is lower than other classic approaches [16]. We still include the tree model as it is an easy and interpretable model. To overcome the non-robustness of a single tree, models that include a collection of trees are built, and will be discussed in section 3.1.4.

3.1.3 K nearest neighbors

The K nearest neighbors (KNN) is a non-parametric method. Unlike parametric methods, such as the previously discussed models, non-parametric methods do not make any explicit assumptions about the underlying model f in $\boldsymbol{Y} = f(\boldsymbol{X})$. This method assumes that observations close to each other belong to the same class. The model classifies an unclassified test case $\boldsymbol{X_{new}}$ by looking at its K nearest neighbors,

and the number K needs to be defined before training the model. In other words, KNN assigns $\boldsymbol{X_{new}}$ to the majority class of the K labeled training observations nearest to $\boldsymbol{X_{new}}$. Which training observations are the K nearest neighbors of $\boldsymbol{X_{new}}$ is determined by their distances to $\boldsymbol{X_{new}}$. The distance between $\boldsymbol{X_{new}}$ and a training observation $\boldsymbol{X_i}$ is defined using the Euclidean distance as

$$d_{new,i} = \sqrt{(x_{new,1} - X_{i,1})^2 + (x_{new,2} - X_{i,2})^2 + \cdots + (x_{new,p} - X_{i,p})^2} \tag{3.8}$$

where $x_{new,1} \cdots x_{new,p}$ denotes the p features of X_{new}, and $X_{i,1} \cdots Xi, p$ denotes the p features of the i^{th} training observation $\boldsymbol{X_i}$. Clearly, if the magnitude of some features are much larger than others, they will dominate $d_{new,i}$. Therefore, to let each feature carry equal importance in the classifier, it is important to standardize data before training a KNN.

One challenge of using KNN is to find the best K, as the number of K directly influences the size of 'neighborhood' and predictions of the model. Setting K to be 1, one can build a super flexible classifier with 100% accuracy on the training data, but it over-fits to the training data and is unlikely to perform well on the test set. However, with a very large K, the KNN loses its advantage of flexibility and is unlikely to perform well.

A method to lesser the importance of setting the best K is to weight the neighbors. The closer a neighbor is to the unclassified case $\boldsymbol{X_{new}}$, the more important it is to the classification prediction of $\boldsymbol{X_{new}}$ thus the higher weight it gets. One of the possible weighting functions is:

$$w_j = \frac{1}{d_j}, \quad d_j \neq 0 \tag{3.9}$$

where w_j is the weight given to the j^{th} nearest neighbor of $\boldsymbol{X_{new}}$, and d_j is the distance between $\boldsymbol{X_{new}}$ and its j^{th} nearest neighbor. The weighted KNN has similar performance compared to the simple majority KNN when sample size is large. When sample size is small, however, weighted KNN might perform better than the other one. For more details bout KNN and weighted KNN, see [3] and [5]

Another challenge of using KNN is its sensitivity to the dimensionality of data. As it is a distance-based method, with the increase of the dimensions, the sum of squared distances of all dimensions becomes similar. As a result, the difference of the distances of a query observation to its nearest and farthest data points becomes negligible. It is not recommended to use the method when the number of observations is not comparable to the exponential of the number of dimensions, or the dimensionality is bigger than about a dozen [2]. Nonetheless, the sensitivity of KNN to dimensionality actually helps us to check if the selected subset of features is informative, as the 'nearest neighbors' are still useful and meaningful when the implicit underlying dimensionality of the data is low.

3.1.4 Tree ensembles

Instead of modeling by a single tree, tree ensembles use multiple trees, and each tree is called a base learner. These base learners' structures are usually very simple and shallow, but together they can form a robust model. These base learners form a committee of predictors, whose weighted prediction becomes the output of the ensemble model. As for the case of classification, the weighted prediction is the majority vote of the base learners. Tree ensembles output feature importance scores as an indicator of how useful each feature is in building all the base learners. For each feature, its importance is calculated as the average of every base learner's Gini index (3.7) decrease through split(s) over this feature.

Various tree ensembles differ in how they build the base learners.

Random Forest

When building a base learner for a Random Forest (RF), we randomly sample with replacement n_{train} observations from the training set, where n_{train} stands for the number of observations in the training set. The difference of an RF base learner and the classification tree we introduced in section 3.1.2 is that when choosing the best feature to split on, RF base learner is only allowed to consider a random subset of all features. A subset of m features is randomly sampled at each split from the full set of p features, and typically we set $m \approx \sqrt{p}$ [16]. The randomness added in RF helps to reduce estimation variances of the model. RF is parallel ensembles, as the learning of any base learner is independent from others.

Extra Random Trees

Compared to RF, Extra Random Trees (ERT) adds extra randomness to the tree ensembles. At each split, not only the candidate feature subset to be considered, but also the cut-point of the chosen feature is randomly picked [7]. The design is motivated by the observations that both the chosen feature at a particular node, and the chosen cut-point of the feature depend strongly on the specific learning sample [8]. Same as RF, ERT is parallel ensembles, and uses added randomness to prevent overfitting to the training samples.

Boosted Trees

In contrast with RF and ERT, boosted tress are sequential ensembles. The idea is to let a new base learner find information not yet learned by its predecessors. There are different implementations of boosted trees, and gradient boosting is used in the thesis. Gradient boosting builds a new base learner upon residuals from the previously grown trees. As base learners of Boosted Trees takes into account the previously grown trees, it is usually sufficient to build smaller trees compared to RF or ERT. In fact, using bigger base trees may easily result in overfitting to the training data, as Boosted Trees continuously add new base learners to correct for previous prediction mistakes made on the training set. Base trees with $depth = 1$

often work well [16].

3.2 Feature Selection

As said in the introduction, the goal of the project is to find the relevant features that influence the performance of different pairing optimization algorithms. If our goal were to come up with best classifier, we may well use all features available. However, our intention is to improve understandings of the pairing problem and the relation between problem features and algorithm performance. Therefore, we want to exclude as many irrelevant and redundant features as possible. We have both limited prior knowledge about what features would be useful and limited data of the pairing problems. Moreover, it is unlikely that all collected features are relevant.

Feature selection is defined as 'a process of choosing a subset of original features so that the feature space is optimally reduced according to a certain evaluation criterion' [40]. For a classification problem, the evaluation criterion is the ability to discriminate samples of different classes [35]. It is a data preprocessing step that aims to eliminate as many irrelevant and redundant features as possible before the training of classification models [21]. Ideally, modeling algorithms should only take in the subset of features to build as simple a model as possible but performs the best on the available data [17]. Therefore, selecting the relevant features is a necessary, and also commonly deployed method to improve the prediction performance of a model [23].

In general, there are three kinds of feature selection methods: filter (univariate), wrapper and embedded [35].

3.2.1 Filter

Filter methods work like a filter, as features are 'filtered' by measurements of the general characteristics of the training data. Features that pass the picked standards are selected. One example of general characteristics of data is variability. Most filter methods score the features one by one, and you can either set a criterion where features with scores higher than the criterion get selected, or you can define a number k, where the top k features ranked by scores are selected. Feature selection with filter methods is independent from the training of modeling algorithms. Selected features are then fed into the learning algorithm. The filter method that is implemented in this thesis is ANOVA F score, and it is formulated as shown in equation (3.10) below:

$$F = \frac{\text{between-group variability}}{\text{within-group variability}} = \frac{\sum_{i=1}^{K} n_i (\bar{X}_i - \bar{X})^2 / (K-1)}{\sum_{i=1}^{K} \sum_{j=1}^{n_i} (X_{ij} - \bar{X}_i)^2 / (N-K)} \tag{3.10}$$

where K is the number of classes of the response variable, n_i is the number of observations in the i^th class, $\bar{X}_i$ is the mean of the i^th group of the feature being evaluated, $\bar{X}$ denotes the overall mean of this feature, X_{ij} denotes the j^th observation in the i^th class, and N denotes the total data size. The higher the ANOVA F score, the less likely that the group means are the same, and thus the more distinguish the corresponding feature.

As filter methods evaluate features individually, they cannot see which features do not add information on other features, which means they cannot eliminate redundant features. Moreover, feature selection with filter methods do not involve the training of classifiers, thus the bias of the filter cannot interact with the bias inherent in a classifier. Feature selection process must consider the impacts of selected features on the performance of a specific classifier to achieve the best classification accuracy [17]. Nevertheless, filter methods are fast to implement, and thus it is still a common practice.

3.2.2 Wrapper

Feature selection with filter methods is independent from the training of classifiers. In contrast with the filter, the wrapper methods 'wraps' feature selections around a classifier. One needs to specify a classifier prior to the wrapper's search of feature subsets. After the classifier is defined, the wrapper algorithm searches for a series of different candidate feature subsets following a search strategy, each of which is then fed into the classifier and evaluated by the classifier's performance. The feature subset that leads to the best classifier performance is the chosen subset by the wrapper.

With p features, there are 2^p possible candidate feature subsets. To make the search computationally feasible, we can use greedy search strategies. The greedy search can work in two directions: forward selection or backward elimination. With forward selection, one starts with no feature and keep adding one feature from the remaining features at each step. With backward elimination, one starts with the full set and keep removing one feature at every step. The process stops when it reaches the desired size of features. The predefined classifier's performance is used to decide which feature to add or remove. With forward selection, we try to add one feature to the current classifier, and the feature that leads to most significant model performance improvement is added to the set of selected features. As for backward elimination, we try to remove one feature from the current model, and the one feature that has the least impact on model performance gets removed. Both are greedy search algorithms as they only look at current situation. The advantage of going backward from the full feature set is that it is easier to capture interacting features [19], although it is computationally more expensive as models are always trained with larger sets than going forward.

There are both advantages and disadvantages of using a wrapper. As features

are evaluated by the learning algorithm itself, in general better feature subsets are selected than the filter approach [13]. The downside is inevitable: the features selected are favorably biased to the predefined classifier [35]. In addition, even with the help of greedy algorithms, the wrapper method is still computationally expensive [37], for one has to train several models to decide which feature to add or remove at each step until the desired feature size is reached.

3.2.3 Embedded

The wrapper methods have to come up with a series of candidate feature subsets prior to feeding the subsets to a predefined classifier and evaluating them. For the wrapper, the search of feature subsets and the training of classifiers are still independent processes. In contrast, in an integrated way, embedded methods simultaneously select features and model the relation between features and the response. The integration of a classifier and feature selection requires a specialized integrated implementation. For the embedded, feature selection is a byproduct of the embedded models. Feature importance is directly computed and updated while building the embedded models. The embedded methods need specialized implementation, but are more efficient than the wrapper as feature selection and modeling are done simultaneously.

We implement two kinds of embedded methods in the thesis. One of them is tree-based classifiers, including Random Forest and Extra Random Trees. A byproduct of these classifiers is ranked feature importance. One can directly select the top k important features after the classifier is built. The other is to use the magnitude of parameters of a regularized linear classifier as the measurement of feature importance [26]. Features with very small parameter have marginal impact on the calculation of the linear function, and thus can be removed.

For the regularized linear classifier, we implement the Logistic Lasso regression (LLR) model. The LLR adds a penalty term to logistic regression's loss function to regularize estimated parameters. LLR uses l_1 norm, defined as $penalty(\boldsymbol{w}) = \sum_{i=1}^{p} |w_i|$ [35], where p is the dimension of the feature vector $\boldsymbol{w}$, i.e. the number of features. Adding the penalty term to the negative log likelihood of logistic regression, the loss function of LLR is:

$$\text{loss}(\boldsymbol{y}, \boldsymbol{w}, \boldsymbol{X}) = \sum_{i}^{N} \log\left(1 + e^{-y_i(\boldsymbol{X_i w})}\right) + \alpha \sum_{i=1}^{p} |w_i| \tag{3.11}$$

where α controls the impact of the Lasso penalty. If $\alpha = 0$, we are back to logistic regression. If *alpha* is large, coefficients of the features will be zero. The LLR both shrinks parameters of features and encourages sparse solution to achieve feature selection[39].

Embedded models are efficient, as feature selection and model fitting are done simultaneously. Moreover, as the model fitting process moves forward by the crite-

rion of improving classification accuracy, it usually is comparable to the prediction performance of the wrapper [35].

3.3 Low-Dimensional Representation of Data

As our data has over twenty features, a low-dimensional representation of the data is required to visualize the data in a low-dimensional space. Principal Component Analysis (PCA) is a method to represent data in lower dimensions. PCA summarizes information in a data set by computing a series of linear combinations of the original features, and each of such linear combinations is called a component. The first PCA component lies in the direction where the data vary the most in the feature space. The succeeding PCA component is always the linear combination of all features that has the maximal variance but uncorrelated to all previous components [16]. Using the first two or three PCA components, one can visualize the data in the two and three dimensional spaces. One can also compute the ratio of the sum of variance of the first k components to total variance of data to judge how well the components represent variances in data. Feature selection may improve the representation of PCA, as noises and redundancy in data negatively affects their performance.

3.4 Evaluation of Selected Feature Subsets

As the goal of feature selection is to either improve prediction accuracy, or to maintain the prediction accuracy of using all features with only a subset of features [4], prediction accuracy will be our main evaluation criteria of each set of selected features. Prediction accuracy is the proportion of correct predictions that are made by a classifier on a set of observations:

$$\frac{1}{n}\sum_{i=1}^{n} I\left(y_i = \hat{y}_i\right) \tag{3.12}$$

where n is the number of observations, y_i is the true label, $\hat{y}_i$ is the predicted label, and $I(y_i = \hat{y}_i)$ takes the value 1 when the predicted label is the same as the true label and zero otherwise.

Another important requirement of feature selection method is that the distribution of predicted classes given the selected features should be as close as possible to the conditional distribution given the 'true' features [20]. As the relevant features should have high predictive power of class labels, the conditional distribution of predicted classes given these features should be similar to the distribution of the true class labels. Therefore, we will also plot the confusion matrices. Confusion matrices visualizes the agreements and disagreements between the predicted labels

and true labels. An example of a confusion matrix is shown as in the example below. C_i stands for the different classes. Row sums of the confusion matrix give us class distribution of the true labels, while column sums shows class distribution of the predicted labels. Moreover, the top left to bottom right diagonal entries give us agreements between the true and predicted labels, while the off-diagonal entries counts disagreements.

		Predicted Labels		
		C_1	C_2	C_3
True Labels	C_1	7	8	9
	C_2	4	5	6
	C_3	1	2	3

Using both evaluations, we can check whether a feature subset is able to both maintain the class distribution in its predictions and keep prediction accuracy high, or it only favors the majority class.

4

Data: Collection and Exploration

As stated in the introduction, the goal of the project is to find a group of features that are able to discriminate the performance of different pairing optimization algorithms. To pursue such a goal, we collect input features that we believe may represent characteristics of a pairing task. The output is based on the performance of different optimization algorithms on a pairing task, i.e. the best performing algorithm on a case.

This chapter first explains two kinds of input features. It then explains the output variable. The next section introduces how the data was collected. The chapter ends with findings from data exploration.

4.1 Input Features

There are two kinds of features used for this classification task. One kind is features extracted from the connection tables, which are directed acyclic graphs (DAG). A connection table contains the primary computations of all possible followers and predecessors of every flight that an airline is operating over a period of time. It is the input of the optimizer. As shown in Figure 4.1, one can consider each flight as a vertex, and the connections from the flight to all its possible follower flights as edges. In a directed graph, the edges have directions. A flight to its follower flight can be seen as a directed edge, and thus the connection table is a directed graph. Moreover, the graph of a connection table is acyclic, meaning there are no cycles in the graph. We refer to Figure 4.1 to explain what is a directed acyclic graph. In the example graph, there is a directed edge from vertex '0' to vertex '2', which means 2 is reachable from '0'. All vertices that are reachable from node '2' are also reachable from vertex '0', as we can go to '2' from '0'. In contrast, vertex '0' is not reachable from '2', as no vertex that '2' directs to can reach '0'. A DAG is a directed graph without cycles, meaning no vertex can reach itself. As one cannot go back in time, a flight cannot reach itself in the graph.

The other kind is the static input features when we run the optimizer to solve a pairing problem. This kind of features include constraints such as the number of active bases an airline has, whether ground transportation can be used to transfer

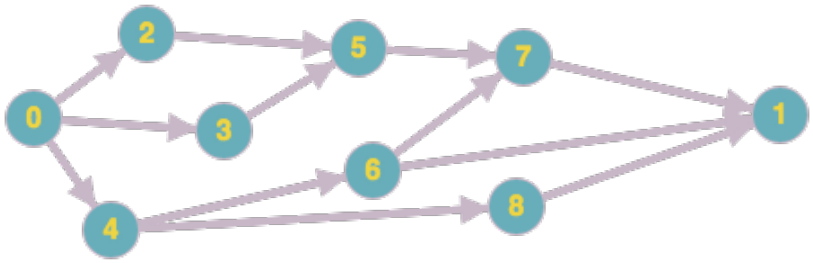

Figure 4.1: The DAG representation of a connection table example

the crew, or how deadheads can be handled.

4.1.1 Features extracted from graphs

A pairing consists of several consecutive flights that start and end at the same base. Not all flights can be the start or end of a pairing, as airlines also operate flights that do not take off or land at their own base(s). We add an artificial 'start of all' and an 'end of all' vertices in the DAGs that we built from the connection tables to keep records of how many of the flights can be the start and end nodes of a pairing. For example, the vertices '0' and '1' in Figure 4.1 denote the 'start of all' and 'end of all' vertices respectively. If a flight can be a start of a pairing, the 'start of all' node is connected to it. Similarly, if a flight can be an end of a pairing, it connects to the 'end of all' node.

Inspired by the article of Smith-Miles et al. (2014), we extract the following features from connection tables:

1. The sizes of the files of the connection tables, as the size of it can represent the complexity and magnitude of a pairing problem. It varies a lot among cases, and its distribution is right-skewed. The smallest case file is only 34 KB, while the biggest is over 1.7 GB.

2. The total number of vertices in the DAG of a connection table: v. The two nodes 'start of all' and 'end of all' are also included in the count.

3. The total number of edges: e. It includes all the edges from the 'start of all' node to the 'end of all' node.

4. Valid start proportion: the number of nodes that can be the start of a pairing divided by the total number of edges.

5. Valid end proportion: the number of nodes that can be the end of a pairing divided by the total number of edges. Together with the valid start proportion, the two features potentially tell us the level of flexibility when setting

up a pairing, as if there are more valid start or end flights, there are more options to group flights into a pairing.

6. Density: the ratio of the number of edges e to the number of possible edges $\frac{v(v-1)}{2}$, which equals to $\frac{2e}{v(v-1)}$. As the graphs are DAGs, between any pair of vertices there are at most one edge, otherwise there is a cycle between the two nodes. Therefore the number of possible edges in our graph is $\binom{v}{2}$.

7. Mean degree: the ratio of number of edges e and number of vertices v, calculated by $\frac{v}{e}$.

8. Standard deviation of vertex degrees: the degree of a vertex is the sum of its in- and out-degree, where in-degree counts how many edges come into the vertex, and out-degrees counts how many edges come out of the vertex.

9. Standard deviations of the vertex out-degrees. The Pearson correlation between the vertex in- and out-degrees is 1, so I only include the standard deviation of the vertex out-degrees.

10. Real mean shortest path lengths: the sum of all shortest paths in steps between any pair of vertices, divided by the number of pairs of vertices where there actually exist a path between them, i.e. the number of pairs of reachable nodes in a graph. It computes the average length of all existing shortest paths in a graph.

11. Theoretical mean shortest path lengths: the sum of all shortest paths in steps between any pair of vertices, divided by the number of possible pairs of vertices $\frac{v(v-1)}{2}$. This measurement also counts in the pairs of nodes that do not have a path between them.

12. Wiener index: the sum of the shortest path in steps between each pair of reachable nodes.

13. Diameter: the maximum of the shortest path lengths in steps between any pair of vertices.

14. Mean betweenness centrality: average fraction of all shortest paths that pass through a given vertex, calculated by mean $\left(\sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}\right)$, where s, t, and v stands for any 3 vertices in the graph that fulfills the requirement that there is a shortest path between s and t. σ_{st} stands for the number of shortest paths between s and t, while $\sigma_{st}(v)$ stands for the number of shortest paths between s and t that pass through v. We compute the mean betweenness centrality both with and without normalization. Without normalization, we compute the mean by dividing the sum by the number of vertices v. With normalization, we multiply the mean by a factor of $\frac{2}{(v-1)(v-2)}$.

15. The standard deviation of betweenness centrality: the standard deviation of

the betweenness centrality of all vertices. We also compute both the non-normalized and normalized version, and we again normalize them by multiplying the values with a factor of $\frac{2}{(v-1)(v-2)}$.

16. Global clustering: measures the extend to which vertices in a graph tend to cluster together, calculated by $\frac{3* \text{ number of triangles}}{\text{number of connected triplets of vertices}}$ The concept comes from undirected graphs so we remove the directions of our DAGs to compute this parameter. For an undirected graph, there is either one edge or no edge between any pair of nodes, labeling whether there is a connection between the two nodes. For 3 vertices in an undirected graph, there are at most 3 edges among them. If all three edges exist, we get a triangle. In contrast, a triplet only requires 2 of the 3 edges. Therefore, a triangle with vertices (x, y, z) corresponds to 3 triplets $\{xy, yz\}$, $\{xz, zy\}$ and $\{yx, xz\}$.

17. Local clustering: another measurement of the tendency of the vertices to cluster together, computed by meanmean $\left(\frac{2L_V}{D_V(D_V-1)}\right)$. L_V stands for the number of links between neighbors of vertex V, and the neighbors of V refer to the vertices that are one step from V and directly connected to V. D_V stands for the degree of vertex V, which is the sum of the in- and out-degree of V. $\frac{D_V(D_V-1)}{2}$ is thus the maximum possible number of links among the neighbors of V. As this method looks at the connectivity of the neighbors of each vertex, it is a 'local' clustering measurement.

In total, we extracted 19 features from the DAG of connection tables. Except for feature 1, all of these features were extracted with the help of the python package NetworkX.

4.1.2 Static features from meta data

The second kind of features, i.e. the static input features can be directly used. These features are mainly categorical features, and there are in total 57 of them. Taking into account that these features are not carefully set case by case (they do not necessarily vary (objectively) over different cases), the less interest from the company in these features, and the rather small size of the data set we currently have, we only include the ones that are deemed potentially interesting by experts at the company. Originally, there were five static features that were interesting to the company. However, only two of these five features actually vary among the cases. As a result, we only include these two features. They are:

1. num_active_bases: the number of active bases owned by the airline company. A pairing must start and end at the same base.

2. num_col_gen_objective_components: The number of components in the cost function.

Together with features collected from the connection tables, we have 21 features.

4.2 The Response

The response variable of this project is categorical, meaning it is a classification task. As the goal is to find features that relate to the best choice of algorithms for different pairing problems, and both the running time and the quality of the solution of a pairing problem must be considered when one decides on the best choice of algorithm for a problem, we do not approach the project as a regression problem.

4.2.1 The classes: three algorithms

As we label a case by the best performing algorithm on the case, the possible classes are the three optimization algorithms run on each case. Details about the algorithms are sensitive business information. Moreover, they are not the focus of this thesis. Therefore we only describe them in general terms. The three algorithms are:

1. The standard (ST, reference) algorithm: the commercial optimizer used by the company.

2. Algorithm AF: compared to the standard algorithm, this algorithm randomly fixes (locks) some of the pairings of a case much earlier. In other words, it stops optimizing some of the pairings in the early stage of the whole optimization process. The purpose of this is to reduce the search space, so that the algorithm can potentially find the final set of all pairings faster. In the cases where the fixed pairings actually do not need any further optimization, the reduced search space would make it easier for the algorithm to find better pairings from the not fixed flights.

3. Algorithm R: adds randomness to the standard algorithm. When the extra randomness is required to find the best solution, or when it helps to save execution time, algorithm R can perform better than the standard algorithm.

4.2.2 Labeling method

To label a case by the best performing algorithm on the case, we need to define what is 'best performance'. Two measurements are collected to evaluate algorithm performance: total cost and execution time.

The total cost measures how well an algorithm solves a pairing problem, for the goal is to minimize the total cost of the cost function. Each pairing problem has its distinct cost function. The cost function contains both real objective costs, such as salary and hotel expenditures, as well as subjective penalties of undesired parings.

One example of the subjective components in the cost is working preference of crew members. Because of the existence of subjective modeling in the cost function, the penalties are different among cases, and costs are not comparable across cases, as different customers prioritize all kinds of requirements differently. However, on the same case (i.e. for the same task from a customer) the formula of cost calculation is the same. Therefore, for each case the total costs of different algorithms are still comparable.

The execution time measures how fast an algorithm solves a pairing problem. All algorithms were ran for all cases under the same conditions (computing resources) using the same stopping criteria. Any run that took over 24 hours on a case was stopped after 24 hours, and its execution time was recorded as 24 hours. In these cases, the minimal total cost achieved within 24 hours is taken as the total cost of the algorithm.

The question is then how to combine these two measurements to decide which algorithm is the best one on a case. The way the company's optimizer outputs the two measurements makes it impossible to fix one and look at the other. The optimizer does not output one optimal solution to a problem, but solves a pairing problem step by step. Each log file keeps track of the step-wise statistics of an algorithm's performance on a case. Therefore, there are the currently achieved cost and the accumulated computation time at each step in the log. As a result, one can find the minimum total cost of the cost function achieved by the algorithm, and the total execution time from the start till the end of a run.

The method that we can use is to find the one that achieves a cost 'equally' low to the lowest cost of any algorithm on the case in the shortest time. As the penalties are subjective, we do not require 'equally low' cost to be exactly the same. Any cost within the 0.5% range of difference of the lowest cost is deemed equally low. The time measurement is total execution time, as this is the time cost for a customer. For each case, we find the lowest cost of all algorithms. If the fastest algorithm, i.e. the algorithm with smallest total execution time, has 'equally low' total cost, then the fastest algorithm is the best algorithm for this case. If the fastest algorithm does not have 'equally low' cost, meaning its solution is more expensive, we compare the second fastest algorithm's cost with the lowest cost. We continue until we can reach the conclusion that no faster algorithm can achieve a cost equally low to the algorithm with lowest cost, and then the algorithm with lowest cost is taken as the best algorithm for this case. If none of the three algorithms could finish within 24 hours for a case, the case is discarded, because we cannot know the execution times and thus cannot compare them. But if some of them took shorter than 24 hour to finish on a case, we can still label the case, as the algorithm that could not finish within 24 hours is simply not the best algorithm on the case. We label all cases using this method.

As you can see, this is a rather complicated way of labeling the cases, making it challenging to classify the cases.

4.3 Data Collection Process

Two kinds of data were collected for the project. The first kind is (historical) customer data of Jeppesen. This data is carefully maintained and used as test suites for the development of the company's optimizer. Among all of Jeppesen's test suites, 118 cases (observations) were provided as data for this project. The other kind was collected from the Official Aviation Guide (OAG) website. The website provides flight data in the world. Four steps were taken to collect useful data from OAG, which are shown as the following:

1. Flight schedules of the first week of May 2014 were collected, as this data was accessible to the company. This data contains flights of 478 airlines. Since the pairing problems of different fleets are solved separately for an airline, the original 478 airlines gave us 1061 cases (observations).

2. Seven different rule sets are applied to each case to turn every single case into seven different cases. The seven rule sets include aviation rules of Civil Aviation Administration of China (CAAC), Civil Aviation Publication (CAP, U.K. Civil Aviation Authority), Directorate General of Civil Aviation (DGCA, India), the European Union (EU), Federal Aviation Regulations (FAR, U.S.), Notices of Proposed Amendment (NPA, European Union Aviation Safety Agency), Notices of Proposed Rule Making (NPRM, U.S.). As the rules decide whether a pairing is valid or not, the expert at Jeppesen believed applying different rule sets to the same case would result in quite different problems for the optimizer.

3. An algorithm was applied to all 1,061 cases to find the bases of all airlines, as a pairing must start and end at a base of the airline.

4. After passing all cases into the company's optimizer, only 62 of the original 1,061 cases were kept. Two factors led to the exclusion of a case. If the optimizer raised any error while solving a case, the case got discarded. The second factor is the size of a case. If a case took too short a time (less than a minute) to be solved by the standard optimizer, i.e. the size of the case too small, the case also got discarded. We end up with 434 cases from the OAG data, as $62 \times 7 = 434$. However, four of the cases could not be solved by any of the three algorithms within 24 hours, resulting in 430 cases in total.

Combining the two kinds of data collected, we got 548 cases in total. For all of the 548 cases, the optimizer is required to come up with a set of pairings that cover all flights of a case.

4.4 Data Exploration

Before the main analysis, one usually explores the data with some simple methods to get basic understandings of the data. As explained in section 4.1 Data Collection Process, seven different rule sets were applied to cases collected from OAG, turning every original case into seven distinct observations. To analyze feasibility of this way of generating data, we compute pairwise correlations of the observations originated from the same case. More specifically, for each original data point X_i, there are seven observations $X_{i,rule_1}$, $X_{i,rule_2}$, ..., $X_{i,rule_7}$ after applying the different rules. Every $X_{i,rule_j}$ has 21 dimensions. The correlations we compute are between each distinct pair of observations of $X_{i,rule_1}$, $X_{i,rule_2}$, ..., $X_{i,rule_7}$, calculated by the formula

$$\rho_{X_{i,rule_j},X_{i,rule_k}} = \frac{\mathrm{E}\left[\left(X_{i,rule_j} - \mu_{X_{i,rule_j}}\right)\left(X_{i,rule_k} - \mu_{X_{i,rule_k}}\right)\right]}{\sigma_{X_{i,rule_j}}\sigma_{X_{i,rule_k}}} \tag{4.1}$$

where $\mu_{X_{i,rule_j}}$ and $\sigma_{X_{i,rule_j}}$ are the mean and standard deviation of the 21 features of the observation $X_{i,rule_j}$, and $\mu_{X_{i,rule_k}}$ and $\sigma_{X_{i,rule_k}}$ are the mean and standard deviation of the 21 features of the observation $X_{i,rule_k}$, and E stands for expectation. When the features almost always take the same values for $X_{i,rule_j}$ and $X_{i,rule_k}$, $\rho_{X_{i,rule_j},X_{i,rule_k}}$ will be almost 1. The data is standardized to mean zero and variance one before we calculate the pairwise correlations.

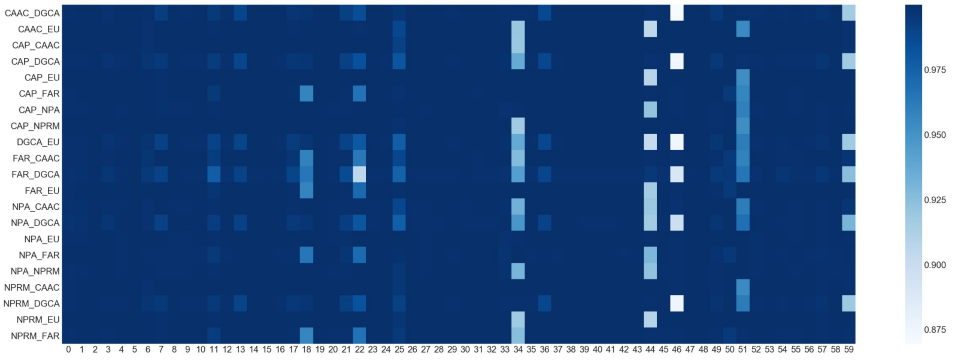

Figure 4.2: Pairwise correlations of data collected from OAG

The values of the pairwise correlations are visualized in Figure 4.2. As we explained in the previous section 'Labeling method', some of the 434 observations cannot be labeled and were discarded. 60 of the original 62 OAG cases have descendents of all seven rule sets, therefore only 60 cases are included here. The vertical dimension denotes the distinct pairs from the seven rule sets, while the horizontal dimension denotes the original case number. All of the correlation values are above 0.875, and the majority of them above 0.975, indicating observations originated from the same case are actually highly correlated. As a result, we cannot treat them as

distinct observations, and can only use data from one of the rule set. We pick data from rule set NPA, as all 62 observations of this rule set can be labeled by our labeling method. We end up with $118 + 62 = 180$ observations in total. Such a small data size makes feature selection necessary from the practical perspective.

There is no missing value in our data. The class distribution is shown in Table 4.1. The classes are relatively balanced.

Classes	Standard	AF	R
Numberofobs	52	77	51

Table 4.1: Class distribution

All visualizations in this section is generated from all 180 observations. Figure 4.3 visualizes the distributions of all 21 features. Every individual plot is the histogram of one feature. For each feature, its entire range of values are first divided into a series of intervals. Then we count how many observations are in each interval. The bars in a histogram visualizes the counts. Most of our features are not symmetrically distributed, either skewed to the left or right. The histograms also show us that value range differs a lot among the features. Therefore, for better performance of classifiers such as KNN and weighted KNN, it is necessary to standardize the data. We standardize the data to mean zero and variance one prior to our analysis.

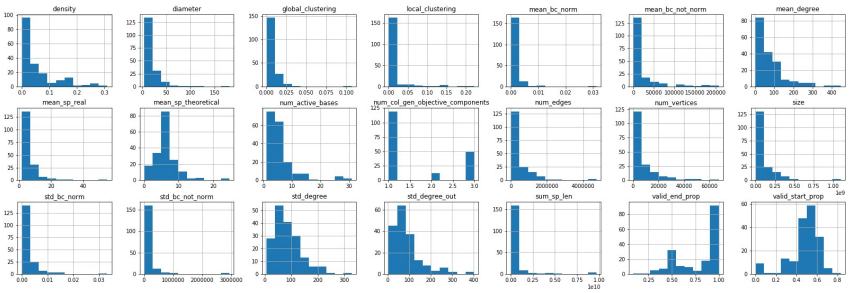

Figure 4.3: Histogram of features

Figure 4.4 includes the box plots of all features against the output classes, visualizing how each feature is distributed within every class. Each box in a individual plot represents all observations from one of the three classes. The vertical direction of the plots denotes values of a feature. A highly informative feature should have non-overlapping value ranges of the different classes. In this case, the boxes would not overlap vertically. Looking at the box plots of our features, we can see that for most of the features, the value spreads of the three classes overlap significantly. Only the features of real mean shortest path lengths, number of edges, number of vertices, size, normalized standard deviation of betweeness centrality, standard deviation of vertex degrees, and standard deviation of the vertex out-degrees seem to have some variations on how the feature values are distributed among different classes.

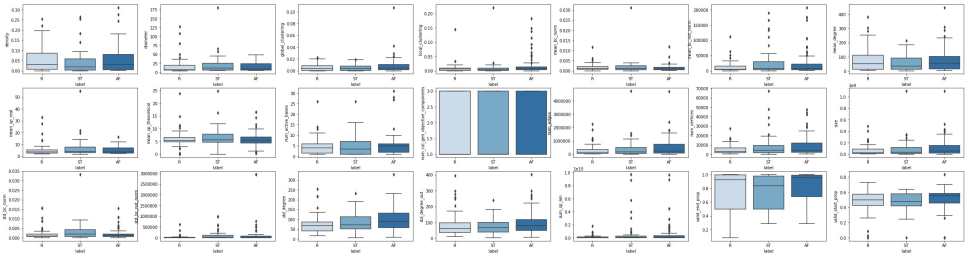

Figure 4.4: Boxplot of features

In the Appendix, we also plotted all distinct pairs of features in Figure A.1. In all of the off-diagonal plots of Figure A.1, two different features A and B are plotted against each other. All of the points are colored by the class labels. In the ideal case where a pair of features are able to classify the observations, we will see three non-overlapping clusters in the plot, with each cluster dominated by one color (class). However, in all of our pair plots, the classes overlap with each other, with no clear division of class clusters.

The plots on the diagonal of Figure A.1 are histograms of the 21 features, which are essentially the same plots as in Figure 4.3. Nevertheless, each bar in the histograms is colored by class labels to visualize class distribution of the observations within the value range of the bar. The colored histograms are consistent with our observations from the box plots in Figure 4.4. For all features, all of the colored bars in the colored histograms are quite 'colorful', with about the same number of observations from every class in each bar. If a feature is highly informative, the bars would be dominated by a single color. In such a case, intervals of the feature values can be matched to the classes, providing valuable information on how to classify observations.

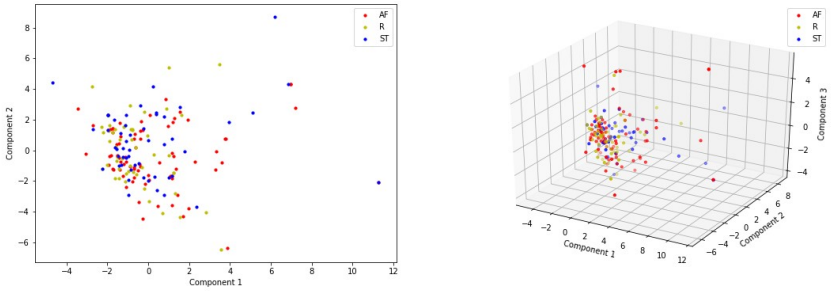

Figure 4.5: Scatter plots of the first two and three PCA components, computed from all features of all observations

We also use PCA to compute low-dimension representation of the 180 observations. The data is standardized prior to the PCA computation, and we compute seven components from the original 21 features. Figure 4.5 includes both the two

and three dimensional PCA representations of our data. The two dimensional representation uses the first two components, while the three dimensional representation uses the first three. All points are colored by class labels. The accumulative variance explained of the data is plotted in Figure A.2 in the Appendix. From Figure A.2, we can see that the first two components can already represent 50% of the total variance in the data, while the first three can represent over 60%. The first two or three components represent the variability in data quite well, however, they are not able to discriminate the cases. Observations from different classes are all clustered together in Figure 4.5. We cannot see any clear division of classes in the figure.

Findings from the above data explorations all emphasize the challenge of our classification task. We do not have any single or pair of features that seem to be informative in classifying the cases. Even the first three PCA components cannot classify the cases at all. Nevertheless, as we mentioned in section 3.3 Feature Selection, irrelevant and correlated features negatively affect the performance of PCA. The next chapter describes the methods we take to try to find an informative subset of features.

5

Methods: Machine Learning and Statistical Analysis

After the data preprocessing steps, including collecting data, extracting features, and labeling cases, we use two ways to analyze the data. The first way is Machine Learning (ML) analysis, and it is the focus of the thesis, as this is the main interest of the company. With ML, we do not pay attention to the interpretation and the statistical significance of the parameters of different models, but rather only apply different algorithms and compare model performance scores. The second way is 'traditional' statistical analysis, and will be discussed in section 5.1. With statistical analysis, we use a basic classification model (logistic regression), and focus on the interpretation of the model and its parameters, as well as model statistics. We include statistical analysis for two reasons. One reason is the data size is quite small. The other is to compare with the performance of ML. All analysis is done in Python, using packages of pandas, numpy, scikit-learn, scipy and statsmodels.

We describe the design of the analysis first, and reasoning of the design comes afterwards. Using machine learning, the analysis processes are designed as shown in Table 5.1. Around 17% of the observations are randomly selected to be the test set. All of the remaining 83% data is used as the training set. We split the data in the way that the class ratios are as similar as possible in the training and test sets.

Steps	Training: 149 obs $ST : AF : R = 43 : 64 : 42$	Test: 31 obs $ST : AF : R = 9 : 13 : 9$
1	Roughly tune hyper-parameters of the feature selection models	
2	Select features	
3	Build classification models using the selected features	
4		Evaluate performance of models on the test set

Table 5.1: Analysis pipeline using machine learning methods

As mentioned in section 3.2 Feature Selection, we use three kinds of feature selection methods: filter, wrapper and embedded. For the filter method, we use the

F score from the ANOVA test, as it tests whether class differences of a feature is statistically significant. The filter method is totally independent from classification modeling, so we do not need to choose a model and tune hyper-parameters for the filter. For the other two, we use Logistic Regression for the wrapper, and Random Forest (RF) and Extra Random Trees (ERT) for the embedded. The wrapper method uses backward feature selection. For the logistic regression model here, we set the hyperparameter that controls the strength of regularization (the hyperparameter C in sklearn) to be 100, and all the rest is kept as the default settings. For RF and ERT, we tune four hyper-parameters, including the number of trees to include in the classifier, maximum depth of a single tree, maximum number of features to be considered when splitting a node of a tree, and the minimum number of observations required to split a node. The five feature selection methods included in this part are summarized in Table 5.2. Logistic Lasso Regression (LLR) is included in the part of statistical analysis.

As the data size is quite small, we use ten-fold cross validation in the feature selection step, in order to estimate the performance of different feature selection methods. Cross-validation (CV) is a model validation method where each fold of the sample becomes the test set for once. When fold i is the test set, the rest of the sample is the training set. After all folds have served as the test set, model performance is summarized across all folds. For our case, we record the predicted labels of every test fold, combining the predictions of all ten folds, and compute the proportion of right predictions, i.e. the accuracy in the end.

$Method_1$	Filter using ANOVA test
$Method_2$	Wrapper using LR
$Method_3$	Wrapper using RF
$Method_4$	Embedded using RF
$Method_5$	Embedded using ERT

Table 5.2: Feature selection methods used in ML analysis

More specifically, we take the following procedures within our ML analysis:

1. Randomly split the data into training and test sets. Standardize the test data to mean zero and variance one. Randomly split the training data into ten cross-validation folds.

2. For each feature selection method $i = 1, 2, 3, 4, 5$:

   For number of features $m = 5, 8, 11, 14, 17, 20$ to be selected:

   (a) For each fold $k = 1, 2, ..., 10$:

      i. Standardize data in fold k to mean zero and variance one. Standardize all data outside fold k to mean zero and variance one.

      ii. If $i \neq 1$ or 2, we tune the hyper-parameters of the model. We create

a grid of hyper-parameters, and for each setting in the grid, we train a classifier using all features and all training observations except the ones from fold k. The hyper-parameter setting that gives the highest classification accuracy on fold k is used as the hyper-parameters for that model.

iii. Use all folds except fold k to select m most important features. For both the filter and the two embedded methods included, the higher the score, the more important a feature is. For the wrapper, we use backward elimination and stops when the feature size is reduced to m.

iv. Use only the selected features to build a classifier, using data from all folds except fold k. As the filter method does not come with a model, we pass the features filtered by this method to a logistic regression.

v. Use the classifier to predict the labels of samples in fold k.

(b) Calculate the overall prediction accuracy of the ten folds.

(c) Compute the number of times a feature is selected by the ten folds. As we have ten folds, and each time the data used to select features is slightly different, we might get different selections from each fold. Features that are repeatedly selected over different fold-splits are considered more important. For the chosen number of features m, all repeatedly selected features by a method are considered the final feature selection result of that method corresponding to m.

We end up with $5 \times 6 = 30$ selected feature subsets, one for each distinct combination of the five feature selection methods and the six numbers of features to be selected. For $Method_i$, we have six selected feature subsets for the six m's . We choose the m that gives the highest CV prediction accuracy, and the corresponding feature subset, as described in step (c) above, is taken as the final selected feature subset by $Method_i$. Note that the size of this final chosen subset for $Method_i$ might differ with the 'best' m, for we take all the repeatedly selected features by the ten folds.

3. Use only the selected features but all training observations to build seven distinct classifiers for each of the five selected feature subsets. The models include KNN, weighted KNN, logistic regression, single classification tree, RF, ERT and Boosted Trees. Default model settings are used in the KNN, weighted KNN, non-regularized LR. For single decision tree, we set maximum depth of the tree to be five, as we neither want the tree to be too shallow or too complicated. RF and ERT use the settings selected from hyper-parameter tuning prior to feature selection. As for Boosted Trees, we set the max depth of a single tree to be one, and the rest settings are the same as default settings. We end up with 35 distinct models.

4. Evaluate performance of the 35 models on the reserved test set. For each of the five feature subset, we compute the mean and standard deviation of the prediction accuracy of the seven models trained on the subset. We also compute the sum of confusion matrices of the seven models for each feature subset. These measurements will show us the predicting power and robustness of each feature subset.

How the data is utilized is visualized in Figure 5.1. For the CV part, the figure shows the case where fold 1 is used as the validation set.

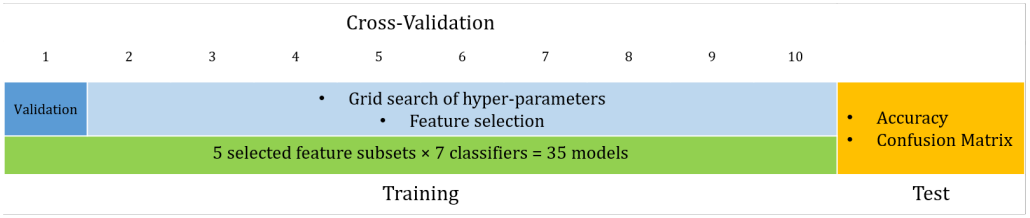

Figure 5.1: Visualization of the split and use of data

As you can see from the above, for each feature selection method, we ultimately keep features repeatedly selected over the ten folds, instead of features selected at once by applying the method to the whole training set. We select features in this way because our data size is really small considering the complexity of our classification problem. The ten CV sets give us ten different samples. By applying a feature selection method ten times to ten different sets, we can get the features that are always selected by a method over different samples. These features are therefore considered more robust and are included in selected feature subset.

Building a set of different classifiers using the same feature subset, and then testing the classifiers' performance on the test set can show us the robustness of a feature subset. Is a feature subset tuned to a specific classifier or always informative across different classifiers? The true important informative features should stay informative regardless of the learning algorithm used.

So why do we choose these seven classifiers? Our classifiers differ from each other, covering parametric and non-parametric models, linear and non-linear models, single and ensemble classifiers, as well as parallel and sequential ensembles. The complexity also varies among our models. LR is the simplest one, as it simplifies reality to a linear combination of input features and only has a fixed number of parameters. In contrast, KNN and the single classification tree can be quite complex models, easily over-fitting to the training data. Simpler models tend to have higher bias, where bias is the difference between the estimated model and the true model. However, there is a bias-variance trade-off in modeling reality with different models. Variance measures how much the estimated model varies across different random sampling, indicating model robustness against the randomness of a sample. Complex models might be able to more accurately capture the real relationship between features and response, but small changes in the training data can result in large changes in estimations. With increasing model complexity or flexibility, usually

model bias decreases faster than the increase of variance at the beginning, but there is a turning point, after which the increased variance cannot be compensated by the decreased bias. We want a model that has both low bias and variance. Implementing a group of models with different levels of complexity will help us to find such a model for our data.

If we were not short of data, it would be better to have a validation set to tune hyper-parameters of the classification models we get from step 3. However, we cannot afford to reserve a separate validation set. Nevertheless, our goal is not to fine-tuning the hyper-parameters to get the most accurate classifier.

5.1 Statistical Analysis

To complete our analysis, we also include statistical analysis after the machine learning analysis mentioned before. As a minor effort of the project, we only implement one model, which is the Logistic Lasso Regression (LLR) discussed in the Theory chapter. There is one parameter to be tuned for the model, i.e. the scaler of the penalty term α. For our ML analysis, we use ten-fold CV to choose the number of features m to be selected for each method. Here, we use ten-fold CV on the training data to choose an optimal α. We experiment with 20 values of $alpha$, and for each α we compute a ten-fold CV. We choose the smallest α that gives the maximum CV accuracy. Larger α means stronger penalty and a smaller model. We do not want a too small model with only a few features, as we want to observe all possibly interesting features. Once again, for each ten-fold CV, we have ten LLR corresponding to the ten folds, and we can plot the number of times a feature is selected by the ten folds. This would show us the repeatedly selected features. After CV, we implement an LLR with the α chosen by CV using all 149 training observations. We interpret the model statistics and some model parameters, and evaluate the training and test accuracies of this LLR. Lastly, we visualize the Lars path of our LLR. Lars path shows the sequence of features entering the model while decreasing α, from the first to the last single feature included in the LLR.

6

Results

6.1 Machine Learning Analysis

6.1.1 Feature selection

This section summarizes the results from applying machine learning methods to our data. As described in section 5.1, we first use ten-fold cross-validation (CV) to select features using all 149 training observations. For each distinct combination of $Method_i$ and number of features to be selected m, we calculate a CV accuracy. For a distinct ten-fold CV, when fold k is used as the validation fold, the remaining nine folds are the training data. Every fold gets to be the validation fold for once. At each step, features are chosen using the nine training folds and are tested on the validation fold. In the end, we get predictions of the 10 distinct validation sets, corresponding to all 149 training observations. The ratio of the number of correct predictions to 149 is the CV accuracy. We adopt five feature selection methods and experiment with six number of features to be selected, and the $5 \times 6 = 30$ CV accuracies are plotted in Figure 6.1.

The solid yellow line in Figure 6.1 shows the prediction accuracy of the dummy classifier, i.e. the ratio of the number of majority class (class AF) observations to 149. In general, all methods fail to exceed the accuracy of the dummy classifier. Only the embedded method using Extra Random Trees (ERT) with m=5, and the wrapper method using Random Forest (RF) with m=11 surpass the dummy classifier's prediction accuracy (0.4295). However, both accuracies are 0.4362 and thus only slightly bigger than the dummy accuracy. The performance of these two settings are not statistically significantly different from the performance of simply classifying all observations to the majority class.

The blue line in Figure 6.1 denotes CV accuracies from applying the Filter method and feeding the selected features to a Logistic Regression classifier. When selecting features, a filter method does not interact with the classifier. As expected, the filter method does give the worst performance, with accuracies below all other methods for most of the time. Generally speaking, the tree-based methods perform better than others, either used in a wrapper or embedded way. The results in

Figure 6.1 also confirm that including more features do not necessarily lead to better performance of models.

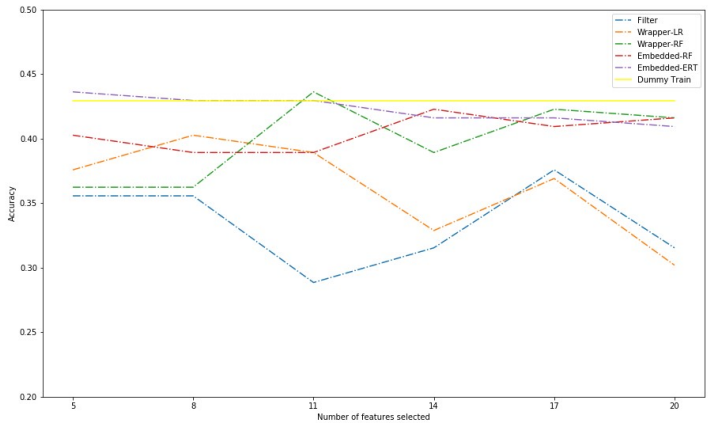

Figure 6.1: CV accuracies for all methods with varying number of features to be selected

Figure 6.2 visualizes the confusion matrices of the CV predictions. For each of the five feature selection method and among the six different m's, we visualize the confusion matrix corresponding to the m that gives the highest CV accuracy. For instance, for the wrapper using LR method, the number of features m that leads to the highest CV prediction accuracy is eight, therefore we visualize the confusion matrix of the CV predictions made by this method when it is required to select eight features. For the filter, the wrapper using logistic regression, the wrapper using RF, the embedded using RF and the embedded using ERT, the numbers of features selected that give the best CV score are 17, 8, 11, 14 and 5 respectively. We also plots all of the nine entries in each of the five confusion matrix. The five subplots share the same color scale, so that the same number is colored by the same darkness. Ideally, our predictions should lie on the top left to bottom right diagonal and have the darkest color here. Figure 6.2 shows that the tree-based methods predict more observations as the majority class (AF), and they perform better compared to other methods only because they tend to classify more observations to the majority class. They are not better at distinguishing the classes. In contrast, the wrapper using logistic regression method actually finds the most R and ST observations. Compared to the wrapper using RF and embedded using RF methods, the wrapper using logistic regression method uses fewer features to achieve this.

As we explained before, for all folds in a ten-fold CV, the splits of training and validation are slightly different. The features are selected based on the training folds. Differences in the training data might lead to differences in selected features. As a result, each fold of the ten folds might select different feature subsets. For a method, we can count how many times a feature is selected by the ten folds. We visualize which features are selected and the number of times every feature is selected by ten-

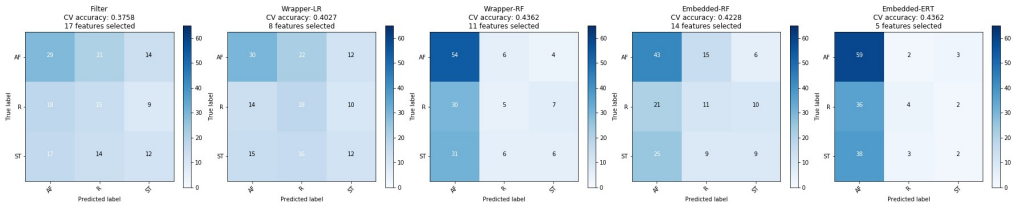

Figure 6.2: Confusion matrices of CV predictions

fold CV in Figure 6.3. For each of the five feature selection method, we visualize the selection counts of the ten-fold CV that has the highest prediction accuracy among the six different m's. For instance, $m = 17$ leads to the highest CV accuracy for the filter method. Selecting 17 features at each fold, the selection counts of the ten folds are visualized in Figure 6.3. The ten folds select 19 of the 21 features more than once, and thus the 19 features become the feature subset selected by the filter method. 12 of the 21 features are selected more than once for the wrapper using logistic regression, 16 for the wrapper using RF, 18 for the embedded using RF and 12 for the embedded using ERT. These repeatedly selected features become the final feature subsets selected by the five feature selection methods.

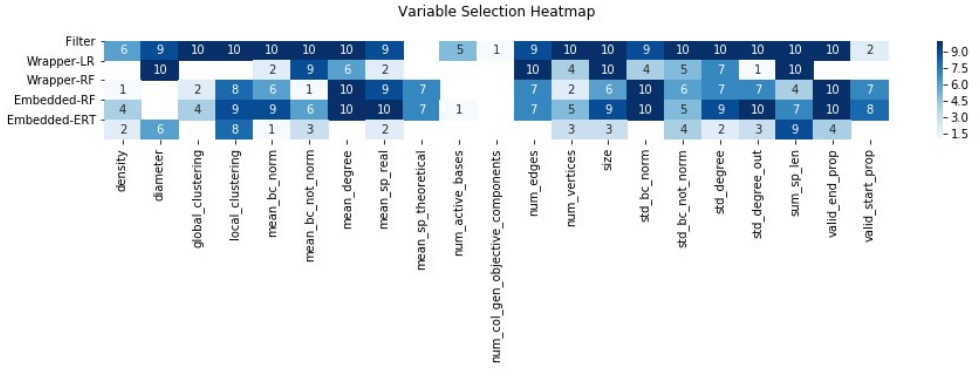

Figure 6.3: Visualization of the number of times the features are selected by the ten folds

The closer a count in Figure 6.3 is to ten, the more certain the corresponding feature selection method is to include this feature in a size m subset of most important features. Compared to other feature selection methods, the embedded method using ERT selects quite different feature subsets over the ten folds. No feature is always selected by the ten folds. In contrast, all other four methods select several features for all ten times. This is consistent with the characteristics of the models. ERT contains more randomness compared to other implemented models. For our data, where the number of observations is small but the classification problem quite complex, the extra randomness in ERT cannot reduce modeling variance, but only adds up to bias and sensitivity to changes in data.

All methods seem to agree that real mean shortest path lengths, number of

vertices, size of the connection table files, non-normalized standard deviation of betweeness centrality, standard deviation of the vertex degrees, and the Wiener index (sum of the shortest paths) are important features, as they all select these features for more than once over the ten CV folds. Four of these six features were detected from the box plot in Figure 4.4 in Chapter 4, they are real mean shortest path lengths, number of vertices, size of connection table files, and standard deviation of vertex degrees.

6.1.2 Performance of classifiers

After getting one selected feature subset for each method, we feed each subset to seven distinct classifiers and end up with $5 \times 7 = 35$ classifiers. The classifiers are all trained on the whole training set of 149 observations, and are tested on the test set of 31 observations. The training and test accuracies of the 35 classifiers are summarized in Table 6.1. For each feature subsets, we calculate the mean and standard deviation of the test accuracies of the seven classifiers, and they are the entries in the two rightmost columns in Table 6.1. These values indicate the robustness of a feature subset. In addition, for each kind of classifier, we compute the mean and standard deviation of the training and test accuracies over the five selected feature subsets, and they are the entries in the two rows at the bottom. This group of values tell us which model fit the data better.

Model	KNN		Weighted KNN		LR		Tree		RF		ERT		Boosting		Test	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Mean	Std
Filter	0.55	0.39	0.99	0.42	0.50	0.26	0.70	0.45	0.74	0.35	0.50	0.42	0.75	0.48	0.40	0.068
$Wrapper - LR$	0.53	0.48	0.99	0.42	0.34	0.19	0.72	0.32	0.73	0.35	0.48	0.35	0.69	0.42	0.36	0.085
$Wrapper - RF$	0.54	0.42	0.99	0.45	0.48	0.23	0.75	0.48	0.72	0.48	0.50	0.35	0.74	0.48	0.41	0.089
$Embedded - RF$	0.55	0.32	0.99	0.35	0.42	0.39	0.75	0.48	0.77	0.35	0.48	0.35	0.76	0.52	0.39	0.068
$Embedded - ERT$	0.55	0.26	0.99	0.23	0.45	0.32	0.67	0.42	0.75	0.35	0.48	0.35	0.74	0.39	0.33	0.064
Mean	0.54	0.37	0.99	0.37	0.44	0.28	0.72	0.43	0.74	0.38	0.49	0.36	0.74	0.46		
Std	0.0080	0.077	0.00	0.079	0.056	0.070	0.031	0.059	0.017	0.052	0.0098	0.028	0.024	0.047		

Table 6.1: Model accuracies on the test set

The highest test accuracy achieved by the 35 models is 0.52. The accuracy of the dummy classifier on the test set is 0.42. As we only have 31 observations in the test set, an accuracy of 0.52 is not enough to confirm this classifier is better than the dummy classifier. In fact, the p value of this classifier being no different from the dummy is 0.18. Therefore, there is a relatively high chance that this classifier is not statistically different from the dummy classifier. In addition, we test 35 models at the same time. If we choose 0.05 as the threshold for statistical significance, we need to divide 0.05 by 35 to adjust for multiple testing (Bonferroni correction, see [9] for more details), for we are testing 35 hypothesis of a model being the same as the dummy classifier at the same time.

Consistent with our findings from the ten-fold cross validation, the feature subset selected by the embedded method using ERT gives the worst mean test accuracy. The wrapper method using RF has the highest mean test accuracy. All the mean test accuracies are below the test accuracy of the dummy classifier.

With regard to the performance of the different kinds of classifiers, the boosted tree modeling has the highest mean test accuracy and thus might be the best kind of classifier for this data. It makes sense as the relation between the features and the classes might be quite complex. Boosted trees base estimators are serial and keep correcting errors from previous estimators, and might be able to catch more information to model the relation between the input and output. Logistic regression model has the lowest mean test prediction accuracy. The simplicity of logistic regression probably is not enough to pick up the complex input-output relation in our data and is not robust against new test data. The weighted KNN models seriously overfit to the training data, as their training accuracies are all 99% and are much higher than their test accuracies. Among the four tree based methods, ERT is least likely to overfit to the training data. ERT has equivalent test accuracies to RF but much lower training accuracies than RF. As a classifier, ERT is not a bad choice, but its extra randomness is not good for selecting features.

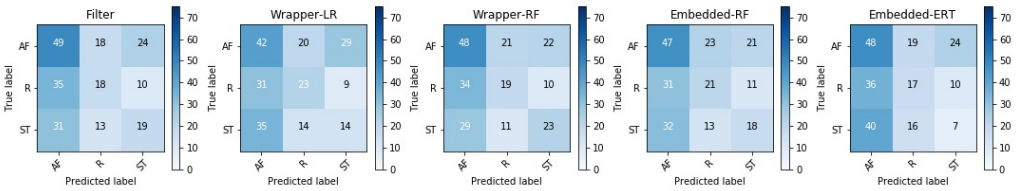

Figure 6.4: Confusion matrices of the test set predictions

Figure 6.4 visualizes the confusion matrices of the predictions on the test set. For each of the five feature subset, we calculate the sum of the seven confusion matrices from the seven different classifiers and plot it in Figure 6.4. Similar to the confusion matrices of the CV predictions, many observations are misclassified to the majority class AF. Distinguishing between the R and ST classes seem to be slightly easier than other pairs of classes, as there are fewer off-diagonal observations in the R/ST blocks.

6.1.3 The best feature subset

This section looks into the feature subset selected by the wrapper method using RF from different aspects, as this subset has the highest mean prediction accuracy over the seven different kinds of classifiers.

Figure 6.5 visualizes the feature importance ranked by an RF classifier. The RF classifier uses the final set of features selected by the wrapper method using RF, and is trained on the whole training set. The features are ranked from the most to least important from left to right. First of all, the feature importance scores do not differ much among the features. We do not have any feature that is significantly more important than others. Secondly, we can see that the top four important features are not commonly agreed important features by the five feature selection methods. In other words, these four features are not repeatedly selected by all five methods.

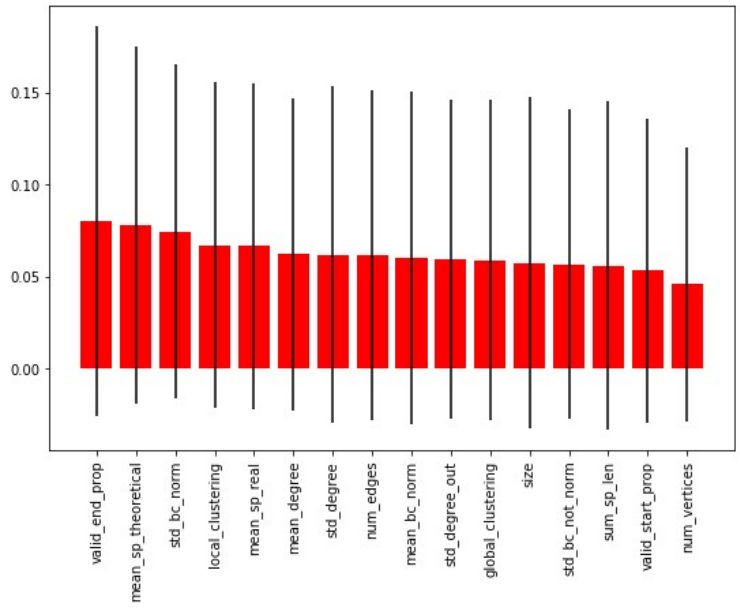

Figure 6.5: Feature importance ranked by the RF model

In contrast, the feature ranks are quite consistent with the feature selection results by the wrapper RF method visualized in Figure 6.3. Table 6.2 compares the feature importance ranked by the RF model, and the number of times a feature is selected by the ten CV folds using the Wrapper RF method. The smaller the rank, the more important a feature is to the RF model. The top half features are all selected more than seven times by the ten CV folds. The top rank feature valid end proportion is selected all ten times by the ten CV folds. Thirdly, the feature importance varies a lot for all features among the base learners of the RF model, as the black line over each red bar denotes the range of $[mean(scores) \pm std(scores)]$.

Features	valid end prop	mean sp theoretical	std bc norm	local clustering	mean sp real	mean degree	std degree	num edges
RF importance rank	1	2	3	4	5	6	7	8
Number of times selected	10	7	10	8	9	10	7	7

Table 6.2: Comparing feature importance indicated by RF feature importance and CV feature selection

In the Appendix, we also visualize a single tree classifier in Figure A.5. The classifier is trained on all 149 training observations using final set of features selected by the wrapper method using RF. The same as the RF model above, the tree classifier also takes the variable valid end proportion as the most important feature, as it is structured at the root of the tree. This variable seems to be able to tell if a case is highly unlikely to be an AF case. After the split at the root, one of the two subsets only contains 4 of the all 64 AF cases in the training set. The other subset still basically maintains the original class distribution. Therefore, according to the tree, if the valid end proportion is smaller than certain value, the case is unlikely to be an AF case. As the data has been standardized, the split point of -1.198 does not give us a sensible threshold directly.

It is not hard to see from Figure A.5 that a single tree can overfit to the training set. After the split at the root, the left branch only has 28 of the 149 observations, but its structure is almost as complex as the right branch. In addition, features, such as the valid end proportion in our case, can occur more than once in a tree's hierarchy, introducing complicated interactions among features. Looking into the changes of class distributions (the value variable in each node) at each depth of the tree, we can see the tree has a hard time distinguishing the classes. All pure end leaves contain less than ten observations, with majority of them containing only one or two observations. Several leaves still have comparable presence of two or even all three classes.

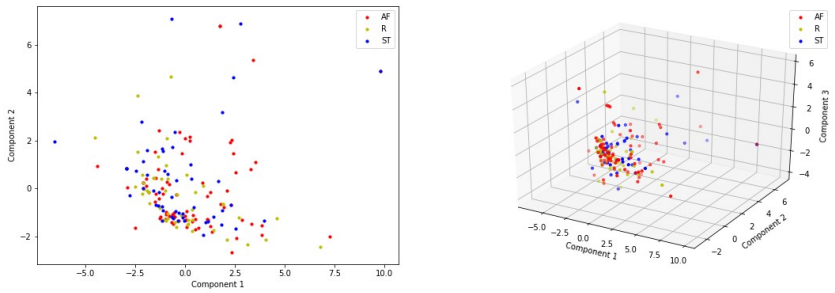

Figure 6.6: Scatter plots of the first two and three PCA components, computed from selected features of all observations

Lastly, in Figure 6.6 we once again visualize the PCA representation of the data. This time we compute five PCA components using only selected features by the wrapper method using RF. As a comparison to the PCA representation prior to feature selection, we also use all 180 observations to derive the components. The accumulative variance explained is visualized in Figure A.3 in the Appendix. Compared to the time when we used all features to compute seven PCA components, we manage to cover as much total variability (over 80%) in the data with fewer components. However, as shown in Figure 6.6, there is still no clear cluster division in the two or three dimensional scatter plots of the data using the first two and three PCA components. Observations from the three classes are still mixed up.

6.2 Statistical Analysis

To complete our analysis and compare with the results from ML, we also analyze the data using statistical methods. The remaining paragraphs of this chapter summarizes results from statistical analysis.

As discussed before, for statistical analysis we use Logistic Lasso Regression (LLR) to select features and model the input and output relationship at the same time, i.e. we use an embedded method. Previously, we used ten-fold CV to choose

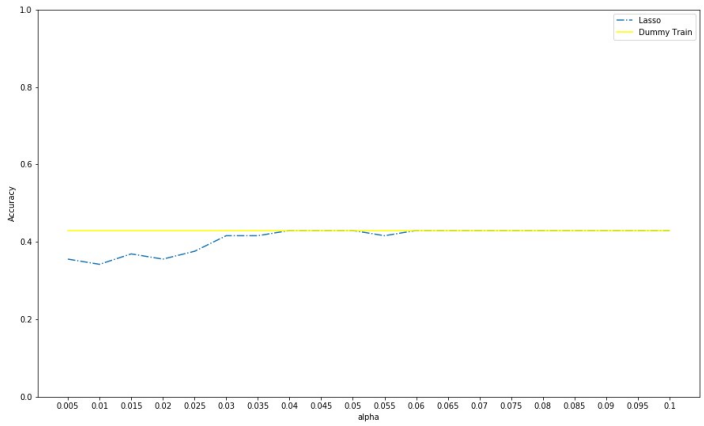

Figure 6.7: CV accuracies for Logistic Lasso regression with varying α

an optimal number of features that should be selected for each method. Here, we use ten-fold CV again, but to choose a satisfactory α, the scalar of the penalty term. The bigger the α, the larger the impact of the Lasso penalty term on the loss function. Figure 6.7 visualizes the change of ten-fold CV accuracies of the Lasso model with varying α using the training set. None of the accurcies is bigger than the accuracy of the dummy classifier, which is denoted by the yellow line in the figure. $\alpha = 0.04$ is the smallest α that leads to the highest CV accuracy. As we do not want to select an alpha that makes an LLR too restrictive, we choose $\alpha = 0.04$.

α	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04	0.045	0.05	0.055	0.06	0.065	0.07	0.075	0.08	0.085	0.09	0.095	0.1
Features	21	16	14	13	8	6	6	6	4	4	2	2	1	1	1	0	0	0	0	0

Table 6.3: Numbers of features selected by Logistic Lasso Regression with varying *alpha*

To have an idea of how many features are actually selected with varying α for our data, we train a separate LLR for each α using all training data, and summarizes the feature subset size of each LLR in Table 6.3. As expected, bigger α means stronger penalty and leads to smaller model. When α is bigger than 0.06, LLR only keeps one or even zero feature in the model. The model's classification accuracy becomes on par with the dummy classifier after excluding all features simply because it turns into a dummy classifier.

Same as before, we visualize the CV feature selection counts. For each of the 20 α, we compute a ten-fold CV. The ten folds create ten LLR models, and we count the number of times a feature is selected by the ten models. All counts are visualized in Figure 6.8. It is not trivial to tell which features are most important directly from Figure 6.8, so we calculate the column sums of the table in Figure 6.8. The sums are shown in Table 6.4. The seven features with a sum of counts equal to or bigger than 80 include number of vertices, non-normalized standard deviation of betweeness centrality, global clustering, the Wiener index (sum of shortest path),

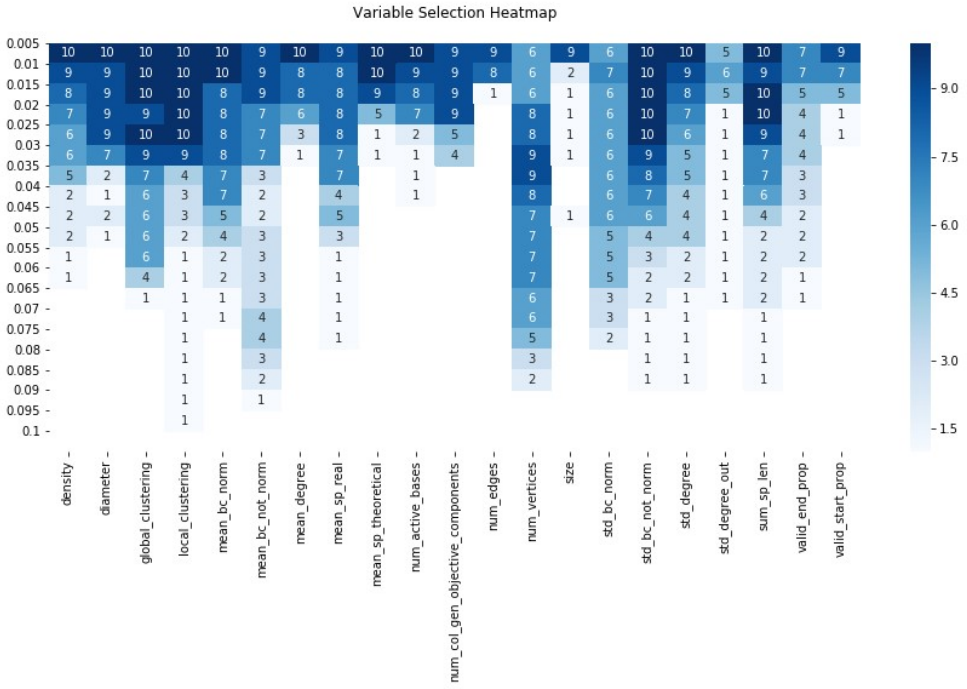

Figure 6.8: Visualization of the number of times the features are selected by the ten folds using LLR

normalized and non-normalized mean betweeness centrality and local clustering. They are ranked by the sum of counts, from 110 to 80. The six commonly agreed important features of the previous five feature selection methods are real mean shortest path lengths, number of vertices, size of the file of the connection table, non-normalized standard deviation of betweeness centrality, standard deviation of the vertex degrees, and the Wiener index. Therefore, the three overlapping features are number of vertices, non-normalized standard deviation of betweeness centrality, and the Wiener index.

Feature	density	diameter	global clustering	local clustering	mean bc norm	mean bc not norm	mean degree
Total counts	59	59	94	80	81	81	36
Feature	mean sp real	mean sp theoretical	active bases	objective components	edges	vertices	size
Total counts	72	36	39	45	18	110	16
Feature	std bc norm	std bc not norm	std degree	std degree out	sum sp len	valid end prop	valid start prop
Total Counts	78	95	71	26	84	45	23

Table 6.4: Sum of the numbers of times a feature is selected with varying *alpha*

Table 6.5 summarizes LLR model parameters and statistics. The LLR model is trained on all training data, and $\alpha = 0.04$. The test accuracy is 0.48, which is the second best accuracy of all 35 classifiers trained previously. Log-likelihood is the value of the log-likelihood of our model. LL-null is the value of the log-likelihood of the model that only include constants, meaning a null model with no features but the constants. As Log-likelihood is bigger than LL-Null, our LLR model is more likely than a null model given the data. The LLR p-value tests the hypothesis of

whether our model is no different from the null model. At a 0.05 significance level, we can reject the hypothesis that our model is no better than the null model.

No.Obs:	149	Log-Likelihood:	-148.89	LL-Null:	-160.71	LLR p-value:	0.023
Df Residuals:	135	Converged:	True				
label=AF		coef	std err	z	P>|z|	[0.025	0.975]
constant		0.3727	0.216	1.725	0.085	-0.051	0.796
density		0.2050	0.259	0.791	0.429	-0.303	0.713
global clustering		0.5638	0.374	1.506	0.132	-0.170	1.298
num vertices		-0.0806	0.288	-0.280	0.780	-0.645	0.484
std bc norm		-0.7178	0.396	-1.813	0.070	-1.494	0.058
std bc not norm		0.3868	0.309	1.251	0.211	-0.219	0.993
std degree		0.0393	0.252	0.156	0.876	-0.455	0.533
label=R		coef	std err	z	P>|z|	[0.025	0.975]
constant		-0.3263	0.341	-0.957	0.339	-0.995	0.342
density		0.0744	0.282	0.264	0.792	-0.479	0.627
global clustering		0.1574	0.391	0.403	0.687	-0.609	0.924
num vertices		-0.1850	0.480	-0.386	0.700	-1.125	0.755
std bc norm		-0.1723	0.213	-0.807	0.419	-0.591	0.246
std bc not norm		-1.6035	1.289	-1.244	0.214	-4.131	0.924
std degree		-0.1911	0.298	-0.641	0.521	-0.775	0.393
		Training accuracy		0.45		Test accuracy	0.48

Table 6.5: Logistic Lasso Regression Results

Besides the constant term, three features of the LLR are not among the seven features with top total counts from ten-fold CV. They are density, normalized standard deviation of betweeness centrality and standard deviation of vertex degrees. The last one of the three is one of the six commonly agreed important features of the previous five feature selection methods, though. In addition, normalized standard deviation of betweeness centrality was detected from the box plots of features in Figure 4.4.

The reference group of our LLR model is ST (standard algorithm), meaning the model compute parameters to differentiate other classes from ST. We use ST as the reference group because ST is the commercial optimizer used by the company. As an example, we explain the two constants to show readers how to interpret the parameters. The constant of AF relative to ST is the log-odds (logit) estimate for comparing AF to ST when the values of all features are zero. When odds is bigger than 1, meaning a class is more likely than the reference class, log odds is positive. As all features of our training data are standardized to mean zero, the constant can be interpreted as the log odds of a case with average values of the six input features being an AF compared to being an ST. This constant is positive, meaning an average case is more likely to be an AF than ST. It makes sense as the majority class is AF. In contrast, the constant of R relative to ST is negative. As we have more ST cases than R, it is natural that an average case is more likely to be an ST than R.

In general, all of the p-values of $P > |z|$ are bigger than 0.05. It means we

cannot reject the hypothesis that all model parameters are zero at 0.05 significance level. If a parameter of a feature is zero, it means the feature has no impact in the model. Besides a constant, only the feature normalized standard deviation of betweeness centrality has a smaller than 0.1 p-value for AF relative to ST. The magnitude of this parameter (absolute value) is also quite big compared to other parameters. Once again, we are conducting multi-testing: testing the statistical significance of 14 model parameters. We should adjust the standard 0.05 through dividing it by 14.

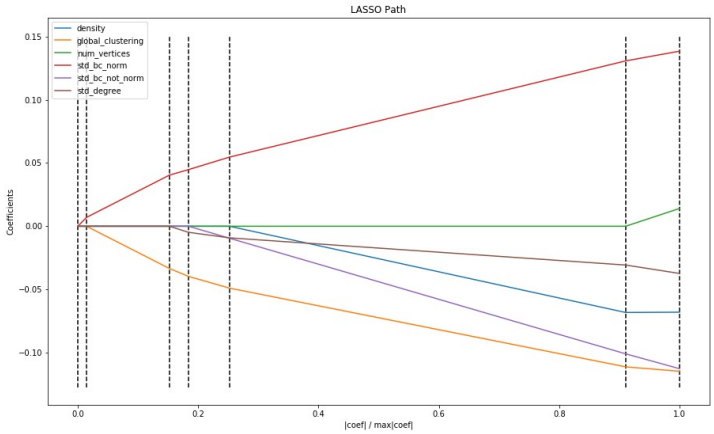

Figure 6.9: Lars path of Logistic Lasso Regression

Lastly, we visualize the Lars path of our LLR in Figure 6.9. As we go from left to right, more and more features are allowed to enter the model. Each dashed vertical line corresponds to a new feature entering the model. A large gap between the dashed lines indicates a region of stability in terms of feature selection, meaning one needs to decrease α a lot to let in a new variable. After normalized standard deviation of betweeness centrality, global clustering, standard deviation of vertex degree and non-normalized standard deviation of betweeness centrality have entered the model, there is a quite large gap. It indicates the model's reluctance to include a new variable after having the four variables.

7

Conclusion and Discussion

This project started with collecting data, extracting features from data and labeling data. Three kinds of feature selection methods were applied, and each selected feature subset was passed to multiple classifiers to test its robustness. The original class distribution and classification results of the data were visualized and compared using PCA. We also compared the results of machine learning with statistical analysis.

7.1 Result Analysis

We knew from the beginning that this project would be challenging, and the results were consistent with our expectations. We applied several feature selection and classification methods, but none of the features seemed to have a high discriminating power, and none of our classifiers significantly outperformed the dummy classifier. The classifiers' test set prediction accuracies were not statistically significantly different from the dummy classifier's accuracy. We were not able to find a solid relation between the considered features and the performance of our pairing algorithms. The low dimensional representation of the data from PCA only showed one cluster of mixed classes, rather than three separate clusters with one for each class.

There are several possible reasons for these results. Firstly, our data may be insufficient. The data is a mixture of carefully maintained (historical) customer data and online data of the flight schedules of the first week of May 2014. On the one hand, we are not sure whether it makes sense to mix these two kinds of data. Having access to this online data does not naturally make it a good complement to the customer data. On the other, the quality of the online data is questionable, as many of them encountered errors while passing through the company's optimizer. Moreover, these error-raising cases were simply discarded. Normally, sub-sampling of data requires random sampling. Sub-sampling by the standard of whether it can be solved by the pairing optimizer without any error is not random. In addition, we only have 180 observations, which is a quite small data set considering the complexity of the task at hand. If there exists a relation between the considered features and

algorithm performance difference, more data might send stronger signals of input and output relations, making the job easier for the feature selection methods and making it possible for these methods to select from more candidate features. More data would also make it possible to use more complicated ML models. More data will also provide us with a larger test set.

Secondly, we used a rather complicated way to classify the cases, i.e. a complicated labeling method. How we classify the cases is what needs to be learned by classifiers. If cases cannot be easily classified, they become challenging to a classifier. However, to decide which is the best algorithm on a case in a meaningful way, we do need to consider both the quality of the solution and the execution time.

Thirdly, the considered features might not be the main factors influencing algorithm performance. We mainly extracted features from the connection tables, consisting of both intuitive and complex features. As we do not have prior knowledge about the connection tables, we only covered relatively common features of directed acyclic graphs (DAG). It might be the case that some special DAG feature is informative for our pairing problem.

Lastly and most importantly, there might not exist a relation between any measurable features of pairing problems and the performance difference of optimization algorithms. Deeper features of a pairing problem may be much more influential, but cannot be measured directly as a feature before solving the problem. However, if we extract features after the problems are partially solved, it will take much time and cannot serve the goal of selecting the best algorithm before actually solving the problem.

7.2 Findings

We did get some interesting findings from our project. Firstly, the seven aviation rule sets are not as different as expected, or at least they do not influence the connection tables as much as expected. Applying the seven rule sets to a case only gave us seven highly correlated observations.

Secondly, according to our labeling method, the default (standard) algorithm currently used by the pairing optimizer did not always perform the best. Most of the times AF (the algorithm with early random locking of some pairings) was the best one on a case. R (the algorithm with extra randomness) seemed to be on par with the standard optimizer, as it was the best algorithm for almost the same number of cases as the standard one. Nevertheless, we used a complex method to rank the performance of different algorithms. The standard algorithm was not always the best because it was not always the fastest, but it might still give the lowest cost of all.

Thirdly, some features are repeatedly selected by different feature selection

methods, and might be interesting to look into. They are number of vertices, standard deviation of betweeness centrality, and the Wiener index (sum of shortest paths). A vertex denotes a flight to be covered, and thus the number of vertices might indicate the scale of the search space of the pairing algorithm. Betweeness centrality measures the centrality of a node in a graph, and thus the standard deviation of them measures the differences of centrality of all nodes. If the standard deviation of betweeness centrality is large, we have both some nodes that many other nodes need to pass through to go further along the shortest path, and some nodes that lie on the periphery. The Wiener index measures the sum of shortest paths, and might be an indicator of the total length of possible pairings.

Besides findings specific to the project, we have also learned some valuable lessons on a more general level. Data is the core of a data analysis project. Collecting data can take a long time. How the data should be collected requires careful design. Before starting a data science project, it is crucial to consider whether and when the data will be available, whether and how well the accessible data is suitable to answer the research question of the project, and to evaluate whether enough data can be collected given the complexity of the project.

With regard to machine learning (ML), clearly it has limitations as with all methods. ML learns from examples in data, and thus data quality is crucial to the success of an ML project. If there are too many noises in the data, ML will learn the noises. If there is not much to be learned in the data, we cannot expect ML to magically find a solid relation between input and output. As with traditional statistical analysis, we need to carefully design analysis procedures and validation of findings. Moreover, ML is not necessarily more favorable than traditional statistical methods, especially when the data size is small. When we have abundant data and a large feature set, ML can save us from manual explorations. However, as shown in our results, when the data size is small, a simple statistical model is comparable to ML models with regard to performance. More importantly, it is easier to interpret. We can directly interpret each feature's impact on the output variable from the parameters of our LLR. This is not possible with KNN and non-trivial with the tree ensembles. ML algorithms should not be treated as black boxes and findings should always be interpreted and reasoned. After all, for this project, we want to understand a relation, not simply to find any relation. Without reasoning, a finding might be worthless, because if we look long enough we will always find some random features that happen to correlate with the output.

7.3 Future work

Before any further attempt, one should consider whether the desired classification can be achieved only using measurable features of pairing problems. If the answer is yes, an obvious next step for this project is to collect more data with quality. One can also explore with how to classify the cases in a more straightfor-

ward way. In addition, one might look into the different rule sets and check if they really are quite similar, as they did not influence the connection tables as much as expected. If the rule sets actually differ, it means they cannot be reflected in the connection tables. In such a case the rule sets can be the new source for interesting features, for the rules should impact on the difficulty of a pairing problem. They were excluded from the project, as the company believed these rules were reflected in the structures of the connection tables. After going through all these considerations, and after more data is collected, one may consider more complex ML models, such as a deep neural network (DNN). Using a DNN can be helpful when the input and output relation is complex, but it requires much more data.

Bibliography

[1] Andersson, E., Housos, E., Kohl, N., and Wedelin, D. (1998) 'Crew pairing optimization', in Yu, G. (eds) *Operations research in the airline industry*. New York: Springer Science + Business Media, pp 228 - 258

[2] Beyer, K., Goldsten, J., Ramakrishnan, R., Shaft, U. (1999) 'When Is "Nearest Neighbor" Meaningful?', *Database Theory – ICDT'99*, pp 217 - 235

[3] Cover, T. and Hart, P. (1967) 'Nearest neighbor pattern classification', *IEEE Transactions on Information Theory*, 13(1), pp 21 - 27, doi: 10.1109/TIT.1967.1053964

[4] Dash, M. and Liu, H. (1997) 'Feature selection for classification', *Intelligent data analysis*, 1 (1-4), pp 131 - 156

[5] Dudani, S. A. (1976) 'The Distance-Weighted k-Nearest-Neighbors Rule', *IEEE Transactions on Systems, Man, and Cybernetics*, 6(4), pp 325 - 327, doi: 10.1109/TSMC.1976. 5408784

[6] Erdogan, G., Haouari M., Matoglu M. O., and Ozener, O. O. (2015) 'Solving a large-scale crew pairing problem', *Journal of the Operational Research Society*, 66(10), pp 1742 - 1754. doi:10.1057/jors.2015.2

[7] Geurts, P., Ernst, D., Wehenkel, L. (2006) 'Extremely Randomized Trees', *Machine Learning*, 63(1), pp 3 - 42. doi: 10.1007/s10994-006-6226-1

[8] Geurts, P. and Wehenkel, L. (2000) 'Investigation and Reduction of Discretization Variance in Decision Tree Induction', *European Conference on Machine Learning*, 1810, pp 162 - 170, doi: 10.1007/3-540-45164-1_17

[9] Goldman, M (2008) Statistics for Bioinformatics. Available at: https://www.stat.berkeley.edu/ mgoldman/Section0402.pdf (Accessed: 05 June 2019)

[10] Gomes CP, Selman B. (1997) 'Algorithm portfolio design: theory vs. practice'. In: Proceedings of UAI-97, pp. 190–7

[11] Guo, H. (2003) *Algorithm Selection for Sorting and Probabilistic Inference: A*

Machine-Learning Approach. PhD thesis, Kansas State University

[12] Guo, H. and Hsu, W. H. (2007) 'A machine learning approach to algorithm selection for NP-hard optimization problems: a case study on the MPE problem', *Annals of Operations Research*, 156, pp 61–82

[13] Hall, M.A. and Smith, L.A. (1999) 'Feature selection for machine learning: comparing a correlation-based filter approach to the wrapper', *Proceedings of the Twelfth International Florida Artificial Intelligence Research Society Conference*, 235, pp 239 - 243

[14] Hastie, T., Tibshirani, R., and Friedman, J. (2009) *The Elements of Statistical Learning*. 2^nd edn. Springer

[15] Hooker, J.N. (1995) 'Testing Heuristics: We Have It All Wrong', *Journal of Heuristics*, 1, pp 33 - 42

[16] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013) *An Introduction to Statistical Learning*. New York: Springer Science+Business Media

[17] John, G.H., Kohavi R. and Pfledger K. (1994) 'Irrelevant Feature and the Subset Selection Problem', in Cohen, W.W. and Hirsh H. (eds) Machine Learning: Proceedings of the Eleventh International Conference. Morgan Kaufmann, San Francisco, CA, pp 121 - 129

[18] Kanda, J., Carvalho, A., Hruschka, E., and Soares, C. (2011) 'Selection of algorithms to solve traveling salesman problems using meta-learning'. *Neural Networks*, 8(3), pp 117 - 128

[19] Kohavi, R. and John, G.H. (1997) 'Wrappers for feature subset selection', *Artificial Intelligence*, 97(1-2), pp 273 - 324

[20] Koller, D. and Sahami, M. (1996) 'Toward optimal feature selection', In: *Proceedings of the Thirteenth International Conference on Machine Learning*. Morgan Kaufman, San Francisco, CA, pp 284 - 292

[21] Liu, Z., Jiang, F., Tian, G., Wang, S., Sato, F., Meltzer, S., and Tan M. (2007) 'Sparse logistic regression with lp penalty for biomarker identification', Statistical Applications in Genetics and Molecular Biology, 6(1), Article 6

[22] Liu, H. and Yu, L. (2005) 'Toward integrating feature selection algorithms for classification and clustering', *IEEE Transactions on Knowledge and Data Engineering*, 17(4), pp 491 - 502

[23] Ma, S. and Huang J. (2008) 'Penalized feature selection and classification in bioinformatics', Briefings in bioinformatics, 9(5), pp 392 - 403

[24] Rice JR (1976). 'The algorithm selection problem', *Advances in Computers*, pp 65–118.

[25] Ramakrishnan N, Rice JR, Houstis EN (2002) 'GAUSS: an online algorithm selection system for numerical quadrature', *Advances in Engineering Software*, 33(1), pp 27–36

[26] Saeys, Y., Inza, I. and Larranaga, P. (2007) 'A review of feature selection techniques in bioinformatics', *Bioinformatics*, 23(19), pp 2507 - 2517

[27] Smith-Miles, K. (2008) 'Towards insightful algorithm selection for optimisation using meta-learning concepts', textitIEEE international joint conference on neural networks, pp 4118–4124

[28] Smith-Miles, K., Baatar, D., Wreford, B., and Lewis, R. (2014) 'Towards objective measures of algorithm performance across instance space', *Computers & Operations Research*, 45, pp 12 - 24. doi: 10.1016/j.cor.2013.11.015

[29] Smith-Miles K., James R., Giffin J., Tu Y. (2009) 'Understanding the relationship between scheduling problem structure and heuristic performance using knowledge discovery', *Lecture notes in computer science*, 5851, pp 89–103

[30] Smith-Miles K. and Lopes L. (2011) 'Generalising algorithm performance in instance space: a timetabling case study', *Lecture notes in computer science*, 6683, pp 524–39

[31] Smith-Miles, K. and Lopes, L. (2012) 'Measuring instance difficulty for combinatorial optimization problems', textitComputers & Operations Research, 39, pp 875 - 889

[32] Smith-Miles K. and van Hemert J. (2010) 'Understanding TSP difficulty by learning from evolved instances', *Lecture notes in computer science*, 6073, pp 266–80

[33] Smith-Miles K. and van Hemert J. 'Discovering the suitability of optimisation algorithms by learning from evolved instances', *Annals of Mathematics and Artificial Intelligence*, doi: 10.1007/s10472-011-9230-5; published online 19th April 2011.

[34] Stutzle T, Fernandes S. (2004) 'New benchmark instances for the QAP and the experimental analysis of algorithms', *Lecture notes in computer science*, 3004, pp 199–209

[35] Tang, J., Alelyani, S., and Liu, H. (2014) 'Feature Selection for Classification: A Review', in Aggarwal, C (eds) Data Classification: Algorithms and Applications. CRC Press, pp 37- 64

[36] Tibshirani, R. (1996) 'Regression shrinkage and selection via the lasso', *Journal of the Royal Statistical Society.* Series B (Methodological), pp 267 - 288

[37] Tuv, E., Borisov, A., Runger, G., and Torkkola, K. (2009) 'Feature selection

with ensembles, artificial variables, and redundancy elimination', *The Journal of Machine Learning Research*, 10, pp 1341 - 1366

[38] Webb, A.R. (2002) 'Feature selection and extraction', in Webb, A.R. *Statistical Pattern Recognition*. John Wiley & Sons, LTD, pp 305 - 344

[39] Xu, Z., Huang, G., Weinberger, K. Q. and Zheng, A. X., (2014) 'Gradient boosted feature selection', in *KDD. ACM*, pp 522 - 531

[40] Yu, L. and Liu, H. (2003) 'Feature selection for high-dimensional data: A fast correlation-based filter solution', in *Proceedings of the Twentieth International Conference on Machine Learning*, Washington DC, pp 856 - 863

[41] Zou, H. and Hastie, T. (2005) 'Regularization and variable selection via the elastic net', *Journal of the Royal Statistics Society: Series B (Statistical Methodology)*, 67(2), pp 301 - 320

A

Appendix 1

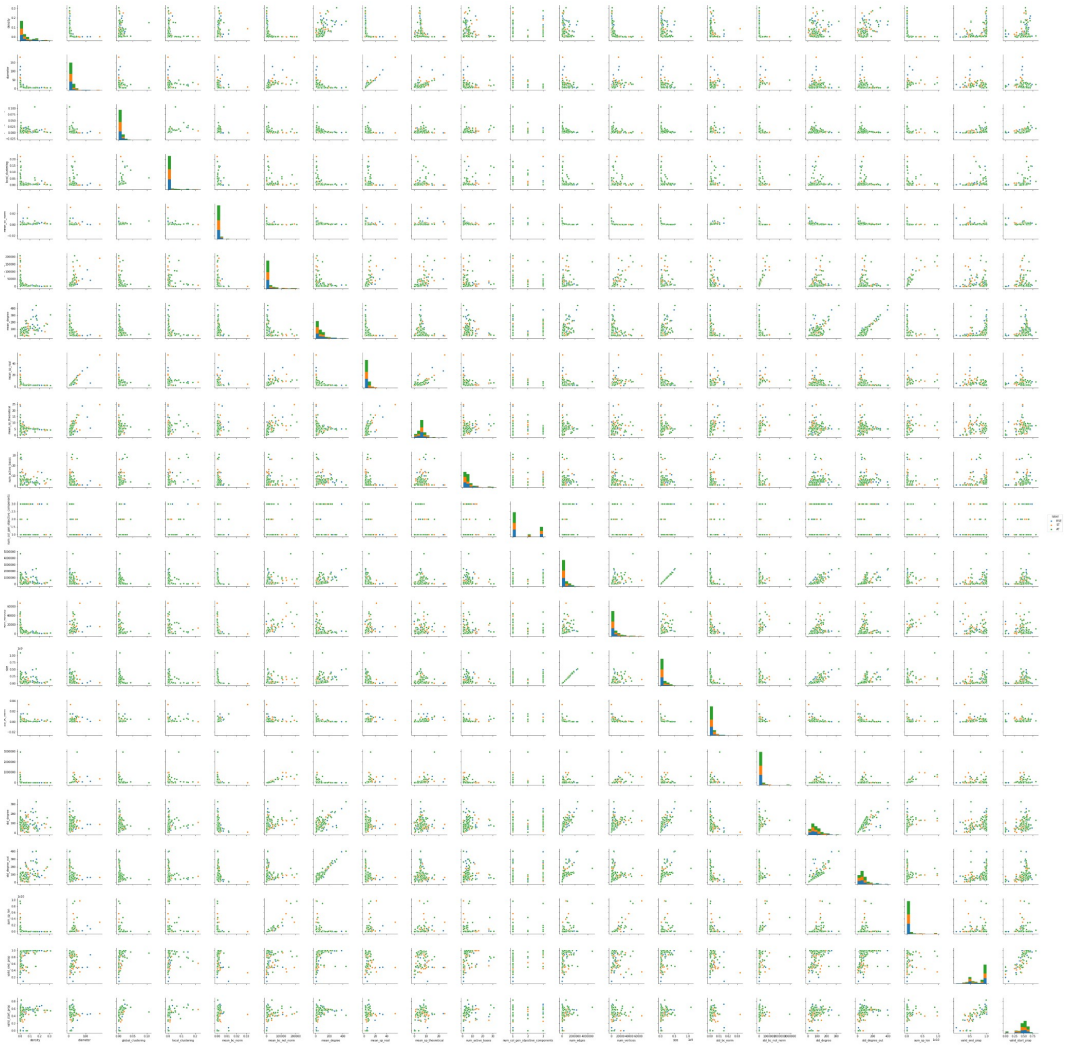

Figure A.1: Pair plots of features

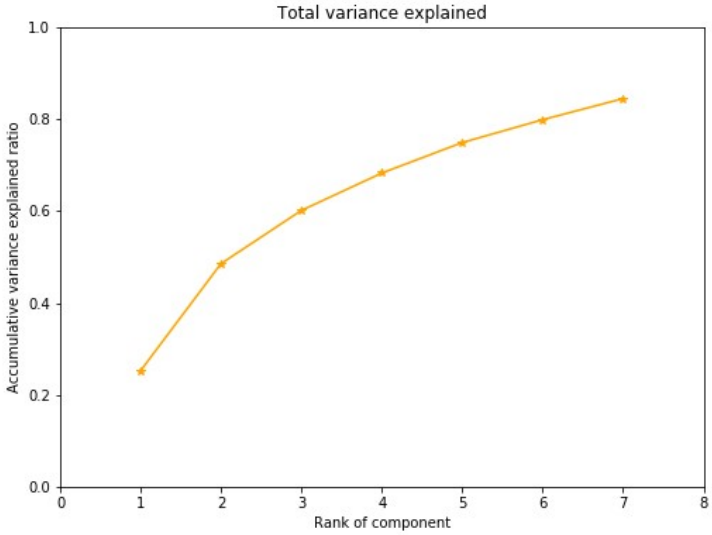

Figure A.2: Accumulative variance explained by the PCA components using all observations and all features

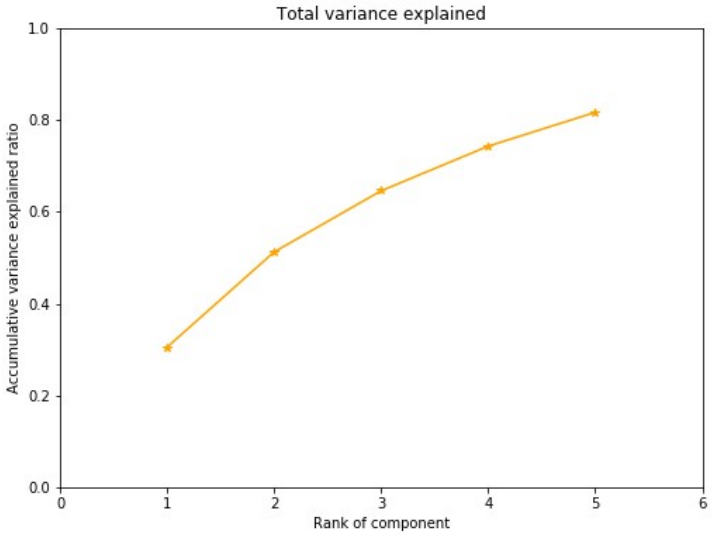

Figure A.3: Accumulative variance explained by the PCA components using all observations and selected subset of features

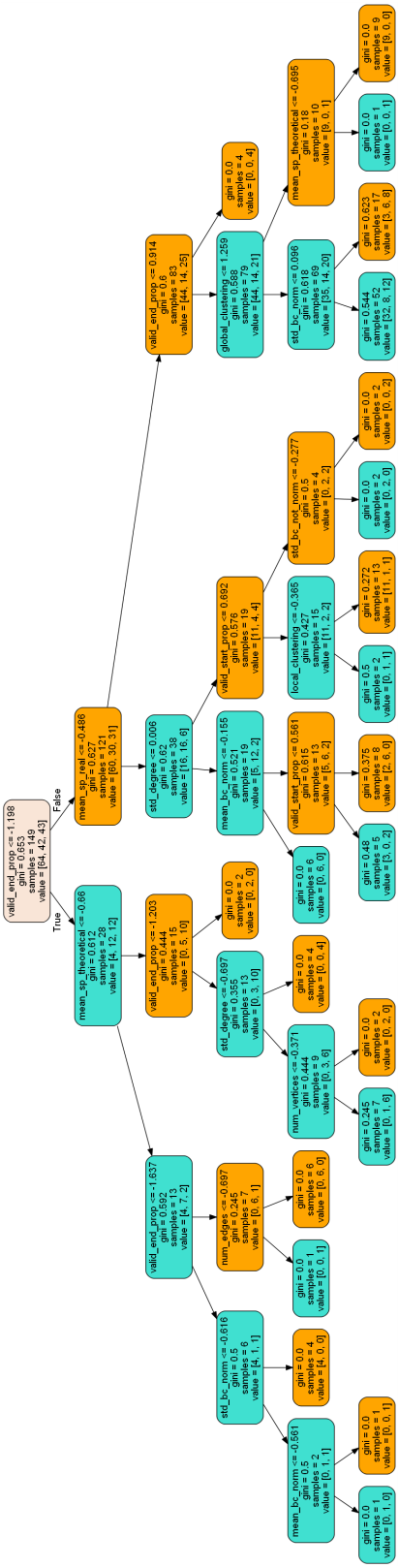

Figure A.4: Visualization of the single decision tree model