No-show Forecast Using Passenger Booking Data

Author:

David Zenkert

davidzenkert@hotmail.com

Lunds Tekniska Högskola

Centre for Mathematical Sciences (Matematikcentrum)
Lund University
Box 118
SE-221 00 LUND
Sweden

Supervisor Matematikcentrum: Umberto Picchini
Supervisors Amadeus IT Group: Olivia Mala, Thomas Fiig, Jan Vilhelmsen
Date: February 24, 2017

Abstract

Amadeus IT Group provide revenue management systems for the airline industry. The concept of overbooking has been known and applied within the industry since the middle of the 20th century, thus playing a large role in a revenue management prospect. The passengers booking data is something that could improve the forecasting of the rate at which passengers don't show up or cancel their respective flights, henceforth referred to as cancellation/no-show rate. This thesis will only address the no-show part but both the concept of cancellations and no-show together are important when overbooking flights optimally. Overbooking too little will result in lost revenues and overbooking too much will result in fees for compensating possibly upset passengers and of course the issue of having to deny boarding to them as well. Therefore, the investigation around how to optimally overbook flights is of importance for Amadeus.

In this thesis, machine learning algorithms are tested with the objective to improve the no-show rates. The revenue management part of this project will not be discussed in great detail.

Acknowledgements

I would like to thank Umberto Picchini at the Centre for Mathematical Sciences for supervising and being both interested aswell as devoted to helping me in this master thesis project at Amadeus.

I would also like to thank Amadeus Scandinavia for letting me do my project at their grounds with all the great colleagues coming with valuable inputs and being interested in what I do. Thank you Olivia Mala and Thomas Fiig for working with me directly, you have been most helpful always answering questions and pushing me in the right direction. A special thanks to Jan Vilhelmsen who has dedicated a lot of his time helping me with everything from my setup at the office and personal advice to finding my first job as an engineer.

Contents

Chapter 1

Introduction

1.1 Background

Amadeus IT Group is a company that provides software systems such as booking and revenue management systems, mainly targeting not only the airline but also the broader travel and tourism industry. Amadeus has its head quarter in Madrid but also large European offices in Nice and Erding.

When you book your flight, the airline keep track of all necessary details regarding your travel in a so called PNR, Passenger Name Record. This record contains a unique code that directly corresponds to your booking. Information contained in the PNRs include information such as your origin airport, your final destination, how much you payed for your ticket, number of people on the booking etc.

Currently, Amadeus calculates no-show rates and cancellation for a given flight based on time series models. On top of that, information in the PNR could contain features describing the no-show behaviour for passengers. Machine learning algorithms are suitable for dealing with multiple inputs where traditional models might be too hard to program yourself or a specific pattern is hard to detect. PNR-based no-show forecasting is something requested by airlines and therefore implemented by Amadeus.

Ideally, the objective is to predict a probability of how likely a passenger is to no-show. This probability is then used as input to a function or a decision rule for overbooking.

The PNRs form a huge data set so only a subset of all the PNRs will be used for modeling. The subset itself is still fairly large, containing about 40 covariates and a few million observations.

1.2 Problem Formulation

Firstly, I've restricted myself to only deal with the no-show rate modeling and prediction, leaving out the cancellation modeling and prediction in this thesis project.

Secondly, the goal is to improve the no-show rate forecasting. Therefore, a baseline model needs to be implemented. The baseline model will work as a benchmark. The algorithms needs to beat this model in order to be considered useful for forecasting no-show rates.

When a baseline model is implemented, the machine learning algorithms needs to be trained on the PNR data and compared with the baseline model.

The analysis is divided into 2 cabins. Cabins in a short-haul flight is the physical line usually drawn by a curtain, dividing economy and business passengers. Cabins in a long-haul flight is usually made up of first class, business and economy. The analysis performed in this thesis are based on the cabins M (Economy) and C (Business).

The main questions in this thesis project are:

- What covariates in the PNR can be used and how? Does some covariates need to be removed or transformed?
- How does the machine learning algorithms work?
- How to implement machine learning algorithms in the statistical software R?
- How to measure which algorithms perform better, what KPIs (Key Performance Indicators) should be used?
- Implement a baseline model and compare the results with the machine learning algorithms.

1.3 Outline

The report starts with a brief explanation of what no-show rates are used for and the concept of overbooking in revenue management. This is followed by a section of data analysis and then machine learning, what it is, how it generally works and then 3 famous machine learning algorithms are explained. The concepts of over/underfitting and cross-validation follows which are important to understand when training a model and when interpreting the results.
Here, a few issues with data sets are discussed along with a few performance metrics/KPIs (Key Performance Indicator).

The 3rd chapter is a walk-through of what was done, how the data was processed and the algorithms implemented.

The last 3 chapters presents the results in forms of plots and KPIs. These are then analysed and thoroughly discussed.

Chapter 2

Theory

2.1 Overbooking

Overbooking occurs when a seller with limited capacity sells more units than he or she has to offer[14]. The reason for overbooking is simply to avoid the lost opportunity for more revenue due to no-showing passengers or cancellations.

Without overbooking, every no-show and many of the cancellations would result in empty seats in the aircraft, a lost opportunity of more revenue. Consider a flight with 100 seats that usually has a higher demand than 100. Also, assume that passengers on this flight on average have a 10 % no-show rate. If the airline never overbooks, it would, on average, depart with 10 empty seats on every flight and at the same time deny reservations to passengers who wanted to be on this flight. This compares to a manufacturer who would plan to run at 90 % capacity, a waste.

Hence, calculating a total booking limit could be done by

$$b = C/\rho, \quad b \geq C \text{ and } \rho \in [0, 1]. \tag{2.1}$$

Where C is the capacity and ρ the show rate, i.e. $1 - NSR$, where NSR is the no-show rate [14].

However, when overbooking too much, there is a penalty called a denied boarding cost. This cost is usually more expensive for the airlines than to have an empty seat [20]. Instead, we want to calculate a very simplified risk-based booking limit that takes into account the denied boarding cost. Let's make a simplified illustration, say the number of passengers that come to the flight are s. They all pay the same price p and that the cost of denied boarding is D with $D > p$.
Both the demand for bookings d and the number of no-shows x are assumed to be unknown and independent [14]. Also define the cumulative distribution

function, (CDF), $F(d)$ which is the probability that the total number of bookings will be less or equal to d. Let $G(x)$ be the probability that the number of no-shows will be less or equal to x. Now, we want to determine for which value of b that maximizes the total expected net revenue. The net revenue is simply given by [14]

$$R = ps - D \cdot max[(s - C), 0]. \tag{2.2}$$

With equation (2.2), we can calculate the expected value of the net revenue which is

$$E[R|b] = p \cdot E[min(b, d) - x] - D \cdot E[max(min(b, d) - x - C, 0)]. \tag{2.3}$$

Now, assume we have set a booking limit $b > C$. The effect of changing the booking limit from b to $b + 1$ would have 3 possible outcomes [14].

1. The demand is $d < b + 1$. Increasing the booking limit would not affect the number of shows and the outcome from increasing the booking limit by 1 would be 0.
2. The demand $d \geq b + 1$ and the number of no-shows is greater than $b - C$. The airline will in this case gain a paying passenger without having to overbook and with the gain p.
3. The demand $d \geq b + 1$ and the number of no-shows is less or equal to $b - C$. This would generate a paying passenger but will also generate a denied boarding to a passenger. This would result in a loss $p - D$ because $D > p$.

Using these 3 outcomes, one can do a probability weighted sum over them. If the weighted sum is larger than 0, one should increase the booking limit b to $b + 1$ because it will generate more expected revenue. Once this is done, you could repeat this process to find out if the booking limit should be increased yet again by one seat. If the weighted sum is smaller than 0, then you shouldn't increase your booking limit.

By using the CDFs for the no-shows G, we could calculate the change in expected revenue that we assume to get when we increase the booking limit from b to $b + 1$ [14].

$$E[R|b + 1] - E[R|b] = [1 - F(b)](G(b - C)(p - D) + [1 - G(b - C)]p) \tag{2.4}$$

Increasing the booking limit by 1 seat will generate more expected revenue if the right hand side in equation (2.4) is positive. Since $1 - F(b)$ is always greater than or equal to 0. This means that when $p - G(b - C)D$ is greater than 0 or equivalently $p/D > G(b - C)$ one could keep increasing the booking limit without losing expected revenue.

This concludes a simple risk-based overbooking algorithm [14]:

1. Begin with $b = C$.

2. if $p/D \leq G(b - C)$, stop. The optimal value for b is the current one.
3. if $p/D > G(b - C)$, set b to $b + 1$ and go to step 2 again.

What this means is that the probability distribution of the no-show rate is important to have in order to use this simplified model. Thus, it is of great importance to the airlines to have an accurate prediction of the no-show rates [17]. A poor model/distribution of the no-shows leads directly to loss of revenue, you either overbook too much or too little.

Even though this is a simplified model, it works closely to the models that airlines use in their booking systems today. Of course, the complexity increases considerably when adding cancellations to this model, the dynamic pricing of tickets, different booking classes, different cabins etc. Anyhow, it gives an understanding for why this project is done and what the no-show forecasting is used for.

Denying a passenger to board costs money, but the passengers reaction and satisfaction is something hard to measure. This is also important to take into account. An airline that frequently denies passengers to board might get a bad reputation and measuring the impact of a bad reputation is also hard.

2.2 Data Analysis

2.2.1 Data Pre-processing

In order to work with the machine learning algorithms, the data needs to be "clean" and data that is wrong would also produce unwanted answers. Notice that there is a difference between removing data that is in some way wrong and removing data that is "unwanted". Removing data that's not wanted could be seen as cheating.

There are several different things that could be referred to as pre-processing the data.

- Converting categorical covariates (Dummy Coding).
- Handling missing values.
- Detecting and handling outliers.

Firstly, covariates might need to be transformed in order to be used since one can't use for instance strings as inputs to machine learning models.
Instead, one could often group and dummy code them instead. Dummy coding simplified is when a categorical covariate has different levels and each level is converted into numerical inputs with levels 0 or 1 [5]. For example, Point of sale which is a covariate with countries as levels is dummy coded:

$$PoS = PoS_{SE}\mathbb{1}_{\{PoS=SE\}} + PoS_{DK}\mathbb{1}_{\{PoS=DK\}} + ... \tag{2.5}$$

where $\mathbb{1}$ is the indicator function that takes the value 1 when the criteria sub scripted is fulfilled. Practically this means extending a categorical variable to so many inputs that there are levels so that when Point of sale = "SE", the $PoS_{SE} = 1$ and all other Point of sale variables are 0. When Point of sale (PoS) is dummy coded, it can be used as an input to the algorithms, whereas the string "SE" or "DK" can't be used as inputs.

Secondly, an issue with real data sets is that there might be information missing. If an element is missing, it will henceforth be referred to as NA (not available). This is a tricky problem and can be dealt with differently depending on the task. Perhaps the most simple solution is to remove the variable containing NAs. This might be a bad idea when you don't have much data or when there are simply too many NAs or if it introduces bias.
Another treatment is to try to impute them [19], i.e. "fill in the gap". There are many different theories around how to impute missing values. One idea is to take the average of the other observations of this covariate and set the NAs to the average. One could also do a linear regression based on other covariates and try to predict the NAs. There are also more advanced methods that tries to impute the most likely value given the other observations that aren't NAs.

In machine learning, some models are actually robust to NAs such as decision trees and gradient boosting. Neural networks are not robust to NAs so to use neural networks the NAs cannot be used as inputs.

Sometimes the data is not missing, it's just that NA might the best label for that specific observation. For instance, if the segment ID is 1, i.e. the first flight the passenger will take, there is no connection time. Therefore, the connection time is set to NA. Not because it's missing but because setting it to 0 implies that there was a connection time and that it was 0.

There is no straight answer to how to detect and handle outliers. The probably most common way is when detecting an outlier is to remove it and treat is as an error in the data. However, it's important to understand the data set so that an important observation isn't removed. If one is trying to model the stock market, a stock market crash is most likely not removed. Another example is if one would find negative values on something that can't be negative such as time or weight. These observations could be treated as outliers and possibly removed unless negative time or weight has a certain meaning in the given data set.

2.2.2 Data Normalization

Especially when working with Neural Networks it is of importance to normalize the data to be able to train the network [16]. It's usually hard enough getting the network to converge and helping the network by scaling the inputs is almost always a good idea. Machine learning methods that don't need normalized data in order to work can still benefit from this approach. Normalizing is important because features with a large range could have a higher impact than those with a smaller range. An example could be changing measurement units from kilometers to meters, which would increase the range and thus have a risk of introducing bias to a model.

One way of scaling down the data is to standardize the data. In mathematical statistics, this means that you subtract the mean of the data (μ_X) and divide by its standard deviation (σ_X) for each data point $x_j \in X$:

$$\hat{x}_j = \frac{x_j - \mu_X}{\sigma_X} \tag{2.6}$$

The standardized data will have a 0 mean and a unit variance.

Another method is to min-max normalize the data. This is a linear mapping of the data to an interval $[0, 1]$. This is done by

$$x'_j = \frac{x_j - m_X}{M_X - m_X} \tag{2.7}$$

Where $M_X = max(X)$ and $m_X = min(X)$. There are more ways of normalizing the data and some methods are more robust to outliers than others.

2.2.3 Skewed Classes

A skewed class is a problem often occuring when talking about machine learning [1]. In our case, the class Boarded/No-show is skewed meaning that one is over/underrepresented. For instance, if you have a training set containing 1000 observations and only 20 of them are of class A and the rest class B, then trying to predict on 100 new unseen observations will most likely result in predicting them all as class B.

Let's say that in the new 100 unseen observations there are 2 of class A (same ratio as in the training set). The predictor will then receive a 98% accuracy by setting all predictions to class B. Depending on what the actual task is, this is a good performance, but when trying to classify for instance no-show passengers, this is bad since the model doesn't predict a single no-show passenger [15].

One needs to be aware of the distribution of the covariates and the output since dealing with a skewed class problem can be tricky. In the given data set, the percentage of no-shows were on average around 2% which makes it a typical skewed class problem.

2.3 Machine Learning

Machine learning is a term used when talking about computer science, mathematical statistics and optimisation that together makes computers able to learn without being explicitly programmed. Machine learning makes it possible for computers to learn from data, detect patterns and from its experience perform tasks like classification, regression and prediction. The more data with good quality and variety it gets fed, i.e. the more experience it can use to train itself will make it better at increasing its own accuracy. The outcome from a machine learning algorithm is usually a measurement such as predicting if a picture contains a cat or a dog (classification), predict housing prices given a number of housing features (regression) and in our case predict the no-show rate as a probability (regression). The aim of the machine learning algorithm is to learn on a set of training examples given both inputs and outputs, then using the obtained model to predict the output given a set of new unseen inputs/observations.

Usually when talking about machine learning you can divide it into two categories [9]:

- **Unsupervised learning** - This means that there are no observed outputs so the algorithm needs to self find patters and structures in the given data set.
- **Supervised learning** - The algorithm has access to observed outputs and can train itself using these. This will be the concept considered in this project.

The popularity of machine learning is growing exponentially since a few years back. This is mainly due to the hardware that has become so much better and can perform costly calculations. It doesn't really depend on new findings within the field. The algorithms are more than a few decades old but haven't been implementable practically until recently. Today - everyone with a dedicated graphical processing unit (GPU) can perform these costly computations. Some examples of where machine learning is being used are [8]:

- Self driving cars
- Computer vision
- Medical diagnosis such as cancer detection
- Voice recognition

This thesis project will only use supervised learning algorithms.

2.3.1 Supervised Learning

A general way of looking at a supervised learning algorithm is considering a data set (x_i, y_i) where $i = 1, 2, .., N$ and x_i being the i:th input and y_i being the i:th output. The objective will be to create a model which can predict outputs for new unseen inputs x and to do it accurately. Depending on the output and how you want to predict it there are several ways to approach this problem.

In supervised learning there are generally two different problems that you can ask your machine learning algorithm to do [9].

- **Classification** - Our outputs belong to a finite set, $y_i \in \{$ Finite set$\}$, but more explanatory might be that the machine learning algorithm is supposed to recognise from the inputs what the output is i.e give it a class or a label. For instance, what animal is there in this picture or what flower is this given the petal lengths and widths. Therefore, predicting a cat as a dog is equally wrong as predicting a cat as a horse. The output is somewhat categorical.
- **Regression** - In opposite to classification, here our outputs $y_i \in \mathbb{R}$. A common example is to predict housing prices given their square meters, number of rooms etc. Thus, if the actual housing price was 500.000$, the prediction 499.999$ is much better than 20$ and not equally wrong as in the classification case.

If you have a classification or a regression type of problem they can be very similar in the way that in classification, the output is predicted as a regression but at the very end mapped onto a class or a label. An example would be if you want to predict a binary class such as "Male"/"Female" or "Dead"/"Alive" etc. If you then fit a regression to the output as a probability of being a "Male" then, given the inputs, a simple divider in the regression i.e. if $p > 0.5 = "Male"$ and $p < 0.5 = "Female"$ could be your binary classifier.

The inputs can also vary, they don't have to be numerical, they can also be ordered categorical variables such as "Bad"/"Ok"/"Good" where each different level of the covariate naturally corresponds to some rang. There are also unordered categorical inputs where the levels perhaps can't be mapped into ranked importance. Examples of this could be Sweden/Denmark/Norway or Blue/Green/Yellow, but not necessarily. If the categorical covariates are ordered or not depends on the given data set and is something that needs to be considered before modeling.

The most common way is to denote X as the inputs and Y as outputs where the capital letters mean that the inputs and outputs doesn't have to be scalars but could be vectors or even matrices. Whereas the lower case letters usually mean a specific observation of a sample x_i, y_i. Since this project is aimed at forecasting a no-show rate, the output y_i will be a scalar and even more precise a probability.

With the data set you would normally form X and Y

$$X = \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \vdots \\ \mathbf{x_N} \end{bmatrix} = \begin{bmatrix} x_{1,1} & \dots & x_{1,k} \\ x_{2,1} & \dots & x_{2,k} \\ \vdots & \dots & \vdots \\ x_{N,1} & \dots & x_{N,k} \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{2.8}$$

Each row represents one observation and in the example provided the data set is of size N i.e number of observations and there are k covariates [23].

When obtaining the data set, one want to divide the data set into three different sets: training, test and a validation set. Unless, one uses a technique called cross-validation. More on this topic in section 2.4.2. Hence,

$$(X, Y) \rightarrow (X_{train}, Y_{train}), (X_{val}, Y_{val}), (X_{test}, Y_{test}).$$

Which also can be seen graphically in the figure below.

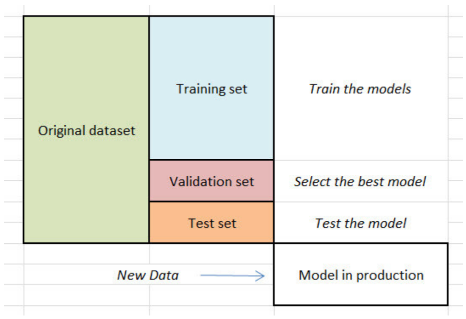


Figure 2.1: Partitioning of the original data set

The procedure most commonly goes as following:

1. Train your model on the training set.
2. Tune the model parameters using the validation set.
3. Test your model on the test set and obtain your accuracy/model performance.

In step 1 above, when training the model, the model takes the inputs x_i from the training set and tries to predict outputs $\hat{y}_i$. These predicted values usually denoted with the hat, $\hat{y}_i$, are compared to the real outputs y_i from the training set. This is done by using a cost function (sometimes denoted as a loss function), in this project denoted by $\mathcal{J}(\hat{y}_i, y_i)$, where the objective is to minimize the cost function w.r.t. our model parameters θ. Our cost function can be seen as a measurement of how wrong our predicted outputs $\hat{y}_i$ was from our real outputs y_i. This is mathematically written as [8]

$$\underset{\theta}{\operatorname{argmin}} \sum_i \mathcal{J}(\hat{y}_i, y_i). \tag{2.9}$$

Usually, the notation is not to have the cost function as a sum of cost functions over each individual sample but to have the entire vector or matrix of outputs in the cost function, all from the training set. Notation wise this is written as $\mathcal{J}(\hat{Y}, Y_{train})$, where $\hat{Y}$ corresponds to the predicted outputs. There are many different cost functions where some of them might be more suitable for each respective task or application but one of the most common one is the sums of squares error (SSE) cost function

$$\mathcal{J} = \sum_i (\hat{y}_i - y_i)^2. \tag{2.10}$$

Another commonly used cost function is the cross entropy function which is mainly used when doing regression with a logistic output, when having a sigmoid activation function (more on this later) and for binary classification. The cross entropy cost function looks like [12]

$$\mathcal{J} = -\frac{1}{N}\sum_i^N (y_i ln(\hat{y}_i) + (1 - y_i) ln(1 - \hat{y}_i)). \tag{2.11}$$

2.3.2 CART - Classification and Regression Trees

Decision trees are a family name for machine learning algorithms that divides the feature space into subsets and gives each subset a constant weight. The models have a gradient structure, like a tree, meaning each subset is reached via branches. They are quite easily understood, they can be compared to the guessing game "20 questions". In this game, one of the players think of an object and the rest of the players get 20 questions to figure out which object he or she is thinking of. The catch is that the other players are only allowed to ask questions that can be answered with "Yes/No". Just like in the game of 20 questions, the tree based algorithms splits the feature set into subsets and assigns these subsets constant weights. This is performed step-wise and the algorithm tries to split on the feature or features that make the most sense to split upon (more on this later). A parallel to the game could be, if you are playing "20 questions" you probably would start by asking a general and wide question like "Is the object dead or alive?" and not a very narrow question such as "Is it Michael Bolton?".

A good way of understanding the CART-algorithm is by an example [9]. Let's consider a regression problem (continuous output Y) and inputs $X = (x_1, x_2)$ where $x_1, x_2 \in [0, 1]$. In this example, only binary splitting is considered, for instance, a divider is added such that $x \leq t$ or $x > t$. In figure 2.2 below, the splitting upon the feature set can be seen to the left. The first step was to add the dividers for $x_1 = t_1$ and $x_1 = t_3$. For x_2 the dividers at $x_1 \leq t_1, t_2$ and $x_1 > t_3, t_4$ was added which formed the 5 different subsets $R_1, .., R_5$. In figure 2.2 to the right, the decision tree that concurs with the splitting upon the features is shown, which is just another graphical way of presenting the splitting.

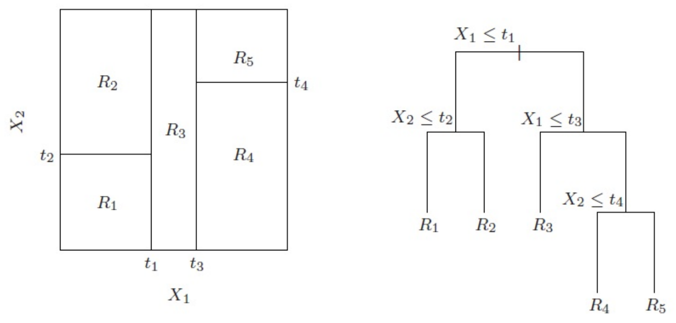


Figure 2.2: Left: splitting on the feature set. Right: coinciding decision tree [9]

As a final step, each region is assigned a constant weight w_m in region R_m

$$\hat{f}(X) = \sum_{m=1}^{5} w_m I_{\{x_1,x_2\}\in R_m}, \tag{2.12}$$

where I is the indicator function that take value 1 if the criteria that's subscripted is fulfilled and 0 otherwise.

Since this is a supervised learner, its goal is to minimize the cost function which we in CART-algorithms define as the SSE seen in equation (2.10) but in this case as

$$\mathcal{J} = (\hat{f}(X) - Y)^2.$$

In this particular case, the best choice of the weights w_m is to set them to the mean of y_i in their respective region

$$\hat{w}_m = \text{mean}(y_i | x_i \in R_m). \tag{2.13}$$

To find the optimal splitting in terms of minimum sums of squares is often not computationally possible. Thus, we use a costly algorithm that starts with all of the data in the training set. Consider a partitioning variable j and split point s and define two halves of the feature set such as

$$R_1(j,s) = \{X|X_j \le s\}, \quad R_2(j,s) = \{X|X_j > s\}$$

The algorithm then searches for the choice of j and s that minimizes the combined SSE over these regions. The sum that the algorithm minimizes is

$$\min_{j,s}\left(\min_{w_1}\sum_{x_i\in R_1}(y_i-w_1)^2+\min_{w_2}\sum_{x_i\in R_2}(y_i-w_2)^2\right).$$

Regardless of the choice of s and j, the best choice for the inner minimization is $\hat{w}_1 = \text{mean}(y_i|x_i \in R_1)$ and $\hat{w}_2 = \text{mean}(y_i|x_i \in R_2)$, just as in equation (2.13).

For each covariate, determining the best splitting point s is a task that's computationally not very costly and therefore, going through all the splitting covariates j is also relative cheap, so obtaining optimal pairs (s, j) is possible. After obtaining a optimal pair (s, j), the set is divided at the obtained splitting point and the method is repeated on the two new subsets. This continues until an optimal tree is found according to a specific model criteria.

Where the splitting occurs is decided but it remains to determine how large the tree should be. This is yet again, as in all statistical modeling, a balance between bias and variance. A large tree would overfit on the training data set and a small tree wouldn't capture the structures in the data set. More on over and underfitting in section 2.4.1.

One naive approach would be to split the tree only on the splits that reduces the SSE by a certain threshold. This is a naive approach, since after a useless split there might come a very significant split. A better approach is to grow a large tree, denoted by T_0 and the tree is grown until it meets a criteria such as minimal number of observations in a node.

Then, a method called cost complexity pruning is used. A tree $T \subset T_0$, is a sub tree that can be any tree obtained from pruning T_0 by removing internal nodes and/or leaf nodes. Leaf nodes are the most outer nodes, just as a tree has leaves as outer contact points. The leaf nodes are represented with m just as each decision region is represented with R_m and let $|T|$ denote the number of leaf nodes in T. As mentioned above, the cost complexity pruning is decided upon the cost complexity criterion seen below [9]

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|. \tag{2.14}$$

Where $N_m = |x_i \in R_m|$ is the number of samples in R_m, $\hat{w}_m$ as in equation (2.13) and $Q_m(T) = \frac{1}{N_m}\sum_{x_i\in R_m}(y_i - \hat{w}_m)^2$. α is the complexity parameter and the objective is to find a subtree $T \subseteq T_0$ that minimizes equation (2.14) for each α, where $\alpha \geq 0$. It can be interpreted from equation (2.14), that when $\alpha = 0$, the solution will be the full grown tree T_0. To find T_α i.e. the tree that minimizes equation (2.14), the tree T_0 is pruned using weakest link pruning.
This means that the internal nodes that provides the least increase in cost is removed and this goes on until only the root remains. This gives a sequence of trees where the optimal sub tree T_α must be listed. [9]. Obtaining the optimal α is done by cross-validation. The optimal $\hat{\alpha}$ is then used to train and prune

the tree which yields the optimal tree $T_{\hat{\alpha}}$. More on cross-validation in the later section 2.4.2.

When doing classification, the method pretty much remains the same with a slight difference in $\hat{f}$ and Q_m. Instead of modeling each region with a constant weight w_m, a class proportion weight $\hat{p}_{mk}$ is instead set to each region R_m such that

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I_{\{y_i = k\}}$$

I.e. the proportion of class k observations in the m :th leaf node. The majority of the class k in node m is then set as the classifier, mathematically:

$$k(m) = \underset{k}{\operatorname{argmax}}\, \hat{p}_{mk}.$$

For Q_m there are three most common choices.

- Misclassification error $\quad 1 - \hat{p}_{mk}(m)$
- Gini index $\quad \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$
- Cross-entropy $\quad -\sum_{k=1}^{K} \hat{p}_{mk} log(\hat{p}_{mk})$

They all have similar functionalities/properties but the advantage with the 2 later ones is that they are differentiable which is preferred when working with numerical data. The later 2 are also more sensitive to changes in node probabilities than the rate of error.

2.3.3 Gradient Tree Boosting

Gradient boosting is an ensemble algorithm meaning that it takes an ensemble of so called "weak" learners and ensembles them into one "strong" learner iteratively. The learners are typically decision trees. In the end the gradient boosting model could be seen as a weighted sum of decision trees. It iteratively fits a very simple model to the data, obtain the residuals and fit another simple model on the residuals from the previous model. This goes on and on until a stopping criteria is met [9].

Consider, in a regression environment, trying to build a model f that predicts values $\hat{y}$ by minimizing the cost function $\mathcal{J}$, such as the SSE.
If the current step for the gradient boosting method is m, where $1 \leq m \leq M$ and M being the total depth of the algorithm. Then, at step m, the boosting model assumes that there is a non-perfect model f_m which it doesn't change but instead improves the model in the next step of the algorithm:

$$f_{m+1} = f_m + h.$$

The logic behind choosing h is quite simple. If we assume that there is a perfect model s.t.

$$f_{m+1} = f_m + h = y \quad \text{or equivalently} \quad h = y - f_m.$$

So in this sense, h is chosen to model the residuals from the previous function. Hence, each step in the algorithm models the residuals from the step before. In gradient tree boosting these functions h are decision trees that tries to model the residuals [13].

The gradient tree boosting model is therefore a sum of decision trees, where each decision tree models the residual from the model before, so the final gradient tree boosting model can be represented as

$$f_M = \sum_{m=1}^{M} T(x; \theta_m), \tag{2.15}$$

where θ_m are each respective trees model parameters. Therefore, at each step m, the method must find the next tree parameters θ by

$$\hat{\theta}_{m+1} = \underset{\theta_{m+1}}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{J}(y_i; f_m(x_i) + T(x_i; \theta_{m+1})). \tag{2.16}$$

The parameters θ_{m+1} must be found for each new model f_{m+1} given the previous model f_m, each set of decision regions $R_{j,m}$ and each constant weight $w_{j,m}$, where j is the index of the decision region. If the regions $R_{j,m}$ were given, then finding the optimal constant weights for each sub region would be easy, just as in the decision trees algorithm [9]. Finding the sub regions $R_{j,m}$ is difficult and the method for finding them varies for each type of problem the algorithm is faced with. For instance, if the cost function $\mathcal{J}$ is a MSE, then the solution is just as easy as for a single decision tree. However, this cost function is not always suitable. The method also differs if the goal is to do regression or classification.

Instead of going into great detail about all the optimization problems around solving equation (2.16), one method for solving it will be discussed, called steepest descent[2].

The idea of steepest descent is again to minimize the cost function $\mathcal{J}$, in the sense of gradient boosting, with respect to the fitted functions f (which could be trees T).

$$\hat{\mathbf{f}} = \underset{\mathbf{f}}{\operatorname{argmin}}\, \mathcal{J}(\mathbf{f}).$$

If we ignore that $\mathbf{f}$ is a sum of trees and instead seen as values of the approximating function $f(x_i)$ on each observation in the data set of length N.

$$\mathbf{f} = \{f(x_1), f(x_2), ..., f(x_N)\}.$$

Again, by solving the minimization of $\mathcal{J}$ numerically as a sum of vectors and obtaining the final model at step M

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m, \quad \mathbf{h}_m \in \mathbb{R}^N,$$

where the first step of the iteration is simply a guess at $\mathbf{f}_0 = \mathbf{h}_0$. Steepest descent obtains the next $\mathbf{h}_{m+1}$ by looking at the gradient of $\mathcal{J}$ and chooses to "walk" in the negative gradients direction since we want to minimize the cost function. This is simply done by

$$\mathbf{h}_m = -\rho_m \mathbf{g}_m,$$

where ρ_m is a scalar (can be seen as the step size) and $\mathbf{g}_m$ is the gradient of $\mathcal{J}(\mathbf{f})$. The gradient is evaluated at the previous step of the iteration $\mathbf{f}_{m-1}$. The $\mathbf{g}_m$ can also be viewed as the pseudo-residuals and are obtained at each iteration $i = 1 : M$ by

$$\mathbf{g}_{im} = \Big(\frac{\partial \mathcal{J}(y_i, f(x_i))}{\partial f(x_i)}\Big)_{f(x_i)=f_{m-1}(x_i)},$$

and the step size is also obtained my minimizing

$$\rho_m = \underset{\rho}{\operatorname{argmin}}\, \mathcal{J}(\mathbf{f}_{m-1} - \rho \mathbf{g}_m).$$

Finally, the next step is then updated as

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m.$$

This is repeated until the step M is reached. This is a costly method since in each step, the gradient for $\mathcal{J}$ is computed and a small step locally is taken. There are ways around this costly method such as stochastic gradient descent which is computationally cheaper which is of great use if the gradient is of larger sizes. Stochastic gradient descent simply randomly shuffles the data set, takes a single observation and performs "gradient descent" on this single observation.

2.3.4 Neural Networks

Neural Networks are probably one of the hottest topics of machine learning and these networks are inspired by how the human brain works.
A human brain has neurons, connected to each other forming a network, where each neuron does a simple computation, hence the name "Neural Network". The Machine learning algorithm Neural Networks mimic this behaviour [8].
There are many different type of neural networks and this thesis will only go into the conceptual most easy to understand, which is a network most commonly

referred to as a Multilayer Perceptron (MLP) or a feed forward neural network. It is called a feed forward network since the information is passed on through the network, it propagates forward. If a network has some sort of feedback connection, they are usually referred to as recurrent neural networks.

The network is usually constructed in the way that a group of neurons in each "step" form a so called layer. A network is said to have an input layer, hidden layers and an output layer.
When talking about deep networks, the only real difference is the number of hidden layers, hence the terms "shallow networks" and "deep networks". A network with one hidden layer is shown below in figure 2.3.

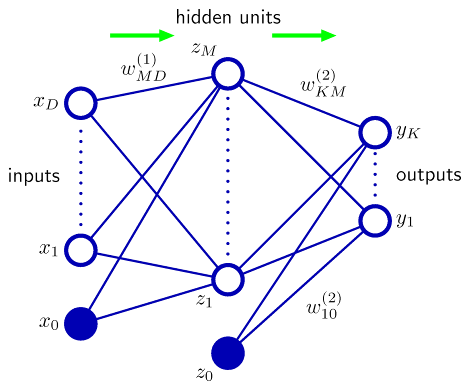


Figure 2.3: MLP with a single hidden layer. Here x_0 and z_0 are bias terms. [6]

The information is passed from the input layer, through the hidden layers and ending up in the output layer. The goal, again is to obtain predicted values $\hat{y}$ in the output layer and to minimize the cost function $\mathcal{J}(y, \hat{y})$. Each line that goes from a neuron to another in figure 2.3 is actually a weight. For each neuron the connecting inputs with their respective weights are fed forward and this algorithm is known as forward propagation. In each layer, there is a constant term known as the bias term. The bias term can be seen as the intercept just like in linear regression that functions as a constant input to each layer. The bias term is usually set to 1. The bias term is also given a weight but is different for each layer, in figure 2.3 the bias term has the subscript 0.

What happens in each neuron is easiest demonstrated by an example. In figure 2.3 above, input to neuron z_1 is a weighted sum of all the inputs from the input layer i.e

$$z_1^{(2)} = \sum_{i=0}^{D} w_{1,i} x_i,$$

Where the superscript (2) meaning the second layer, in this case the hidden layer. The subscripts $1, i$ meaning that the weight represents input i going to neuron 1 in the hidden layer, for this example. In matrix notation, what is fed to the hidden layer neurons is

$$Z^{(2)} = XW^{(1)}, \tag{2.17}$$

where X is the matrix of inputs and W the weights for the input layer. However, the notation Z above, usually referred to as the activation is what is fed to each new layer but is not the output from that same layer. The activation is fed to something called an activation function, a usually non-linear function that transforms the activation for each layer. The activation function produces a non-linear decision rule by using non-linear combinations of the inputs who are multiplied by their respective weights. Commonly used activation functions in MLP are

$$\text{Sigmoid} \quad \sigma(x) = \frac{1}{1+e^{-x}} \quad \text{or} \tag{2.18}$$

$$\text{Hyperbolic Tangent} \quad \sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{2.19}$$

where the Sigmoid transfers the activation to values $[0, 1]$ and the Hyperbolic Tangent transfers them to values in $[-1, 1]$ [8].

Moving forward in the network, the activation in the hidden layer, $Z^{(2)}$ in equation (2.17), is then put into let's say the Sigmoid activation function. This yields the output from the hidden layer, the 2nd layer in figure 2.3, henceforth referred to as A,

$$A^{(2)} = \sigma(Z^{(2)}). \tag{2.20}$$

After obtaining the activation for the output layer, the output from the hidden layer is multiplied with the weights from the output layer,

$$Z^{(3)} = A^{(2)}W^{(2)}. \tag{2.21}$$

Then again, applying the activation function to the activation Z we obtain the output from the output layer

$$\hat{Y} = \sigma(Z^{(3)}). \tag{2.22}$$

This is the forward propagation algorithm and it works pretty much the same regardless of what the network looks like [10] [8].

The weights for the network are usually initialized at random with some sort of constraint, for instance, they could be drawn from a normal distribution. If we add equations (2.17)−(2.22) together, the expression $\hat{Y} = \sigma(\sigma(XW^{(1)})W^{(2)})$ is obtained. When plugged into, let's say a SSE cost function, it looks like

$$\mathcal{J} = (Y - \hat{Y})^2 = (Y - \sigma(\sigma(XW^{(1)})W^{(2)}))^2. \tag{2.23}$$

When talking about training the network, it means adjusting the weights to minimize the cost function $\mathcal{J}$. One can see that trying to minimize equation

(2.23), the only thing that can be altered are the weights W, since we can't change the inputs X or the known outputs Y.

Again, minimizing the cost function in equation (2.23), is an optimization problem usually solved by the back propagation algorithm. Back propagation uses gradient descent or a version of it when training neural networks.

Gradient descent is conceptually easy and works like steepest descent, mentioned in section 2.3.3, and is almost easier visualized in the neural network case. The algorithm tells us to move in the negative direction of the gradient of the cost function $\mathcal{J}$ since we want to minimize it w.r.t the weights. Mathematically, the weights in layer n are updated such that

$$W_{new}^{(n)} = W_{old}^{(n)} - \alpha \nabla \mathcal{J}, \tag{2.24}$$

where the step size α is in neural networks referred to as the learning rate. It is a hyper parameter that can be regularized. A large α has less risk of getting stuck at a local minima instead of the global minima, but also has the increased risk of missing the more precise minima in the optimization process. Therefore, a larger learning rate makes the process go faster and more robust against local minima.

When having a large network or a large data set, taking the entire gradient of all the weights is costly. Therefore, many neural networks are trained using stochastic gradient descent or a version of it. A simple stochastic gradient descent is seen below. Again, instead of using the whole gradient, it takes a single observation and performs "gradient descent" on this randomly drawn observation.

$$W_{new}^{(n)} = W_{old}^{(n)} - \alpha \nabla \mathcal{J}(x_i, y_i). \tag{2.25}$$

This process is then repeated for all the observations in the training set [18].

There are other versions of stochastic gradient descent such as batch gradient descent where a subset of the training set is chosen for each iteration instead of a single observation, which could be seen as a mix between stochastic and normal gradient descent.

The algorithm of back propagation uses gradient descent in each step for updating the weights. Going back to the example in figure 2.3, the back propagation algorithm updates the weights $W^{(1)}$ and $W^{(2)}$ independently at the same time, using equation (2.24), for each iteration in the back propagation algorithm. The weights are updated until a stopping criteria, a performance rate, convergence or a specified number of iterations is met. However, back propagation using gradient descent doesn't guarantee that we will converge and find a good solution, find a solution in a finite number of iterations or find a solution at all [11].

2.4 Model Evaluation

2.4.1 Over- and Underfitting

These two concepts are usually important topics when talking about machine learning [7].
When talking about underfitting, one usually refers to that the model hasn't learned the behaviour in the data set well enough. An opposite to this is overfitting, meaning that the model has learned the behaviour of the training set too well. An overfitted model is unlikely to perform well on new data. Choosing the right complexity of a model is usually a challenging problem since the model should generalize well to new data. A visualization is provided below in figure 3.1.

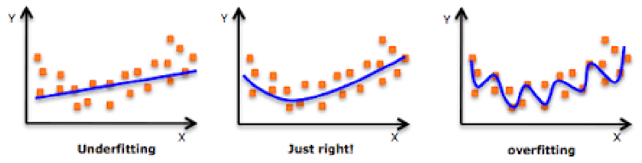


Figure 2.4: Underfitting, "Good model" and overfitting. [21]

A good way of checking if you have over- or underfitted your model is to check the accuracy of the model on both the training and the test set. Generally speaking you have:

- Overfitted - When the the accuracy on the training set is high but the accuracy on the test set is low.
- Underfitted - When the accuracy on both the training set and the test set is low.
- Good fit - When the accuracy on the training set is high and the accuracy on the test set is slightly lower on the test set, but still high.

One of the most common problems in modeling is that you take a very complex model, fit it on your training data and think that you are done since the accuracy on the training set will be very high. This rarely captures the general behaviour of real data, whereas a simpler model usually generalizes better to new data [22].

With four coefficients I can fit an
elephant, and with five I can
make him wiggle his trunk.

John von Neumann

2.4.2 Cross-Validation

Dividing the data set into training, test and validation set might not always be the best solution. One obvious issue with this approach is if the data set is small, then one would preferably want to use as much data as possible for training the model. Another problem might be that when partitioning the data, the distribution of the data might be different in the three subsets. This leads to a group of data is under- or overrepresented when training which leads to errors when forecasting.

An extreme example would be if you try to teach a student to read, without the student actually knowing the alphabet. The book you give the student to learn from, for some reason, doesn't contain certain letters. But when you test the student by giving him a new book containing these for him unseen letters, then, all of a sudden, the student can't read. It this case, it would have been better to give him chapters from both books so that in the learning process, he was exposed to all letters in the alphabet.

A way to work around this problem is to use cross-validation [3]. The data set is now only divided into a training and a test set, with a usually large training set containing maybe 90% of the data. Instead of only training the model once, the model is trained K times on K different training sets which reduces bias. The test set is left untouched since when measuring accuracy, you want the model to predict on unseen data. However, the training set is divided into K equally sized subsets where the subsets are disjoint from each other. The K-fold cross-validation method is [9]:

- For each fold, take the union of $K-1$ subsets as the training set.
- Test on the Kth subset.
- Repeat until all of the K subsets have served as the test set.

Hence, if 5-fold cross validation is used, divide the training data into 5 equally sized subsets. Use subsets $1-4$ to build the model and test on the 5th subset. Then, take subsets $1, 2, 3, 5$ as the training set and validate on subset 4 and so on. This means that each fold will have been used $K-1$ times for training and once for testing. Having used K-fold cross validation, the performance is measured as the average of the prediction accuracy over the K folds. However, using K-fold cross-validation does reduce bias but does increase variance. Usually, 5-fold or 10-fold cross-validation is used.

2.4.3 Performance Metric

The most common way of measuring the performance of a learner is to calculate it's accuracy or error rate. This is simply done by averaging the number of times the classifier was right or wrong. This means that the accuracy is a measurement in the interval $[0, 1]$. This means that if a learner has an accuracy of 1 i.e 100% then it obviously has an error rate of 0%.

When dealing with skewed classes, this performance metric might not be the best way of measuring how well the learner performs. When dealing with binary classification it is very suitable to print a so called confusion matrix. A confusion matrix tells us what was classified as what with the correctly classified outputs are on the diagonal. A confusion matrix looks like

Table 2.1: Confusion Matrix

	Predicted	0	1
True			
0		a	b
1		c	d ,

where a are the number of outputs the classifier correctly classified as 0 and b the number of outputs the classifier incorrectly classified as 1 that actually were of class 0. The same reasoning goes for c and d. These letters $a - d$ are usually referred to as seen in table 2.2 below.

Table 2.2: True and false positives/negatives

a: true negatives	b: false negatives
c: false positives	d: true positives

From the confusion matrix (table 2.1), two easy but useful performance metrics are Recall and Precision [4]. Recall is simply the proportion of true positives out of the set of true positives and false negatives i.e

$$Recall = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}, \tag{2.26}$$

Which can be interpreted as how well the classifier predict class 1 given that it is of class 1.

Precision on the other hand is the the proportion of true positives out of the set of true positives and false positives

$$Precision = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}, \tag{2.27}$$

Which can be interpreted as what proportion of what was classified as class 1, was classified correctly.

The following ways of comparing results of predictions are later used as our KPIs:

- the mean error, ME, often referred to as the bias.

$$ME = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i) \tag{2.28}$$

- The root mean square error, $RMSE$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} \tag{2.29}$$

- The mean absolute error, MAE

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| \tag{2.30}$$

- Lastly, the mean absolute scaled error, $MASE$

$$MASE = \frac{MAE}{\frac{1}{n-1}\sum_{i=2}^{n}|y_i - y_{i-1}|} \tag{2.31}$$

All these different error measurements have their advantages and disadvantages. Only a few of these will later on be discussed in the discussion chapter.

Chapter 3

Methodology

3.1 Data Preparation

As an initial approach, only one segment in the data set is investigated. Let's call it segment AAA-BBB and all flights from AAA to BBB during one year is filtered out as the new data set. The models and comparisons will be based on this subset. The covariates being used can be found in the Appendix A.

Another initial approach is that the subset is divided into a training and a test set or more precisely a test month where the 2 nearby months to the test month are excluded from the training set. Due to some flights might occur in both the train and the test set if these are included. This can be seen as a 1/12 of a cross validation on the full subset of the segment AAA-BBB.

Looking at every flight for this airline throughout a year would be too much as an initial analysis and the goal is to see if machine learning algorithms can beat the baseline model. If the algorithms can't beat the baseline model on the segment AAA-BBB trained on 9 months of data on a test month then it doesn't look very promising. The first test month is chosen so that it has a specific behaviour with some abnormal events occuring, e.g Christmas or longer public holidays.

In this thesis project, a data file containing a few million PNR's is obtained, all from the same airline. These are extracted from a BOF (Bulk Origin & destination File) and before implementing models the data needs to be analysed and understood. The covariates used or versions of the used covariates are seen in appendix *A*.

The data is plotted in different groupings in the software Tableau for basic analysis and comparisons. Some of these plots are seen later in the result section.

As an example, a few lines of data are generated. This is for the reader to get a better grip of what more precisely was used when modelling. All data in the Appendix B are fabricated for illustration purpose. The data in Appendix B is fabricated due to company secrecy. All the covariates explained in Appendix A are then used as inputs to the machine learning algorithms when trying to model the no-show rate, usually denoted in the R code as NSR. The only ones not used from Apendix A are the following:

- Origin board and off point.
- List of Airports.
- Booking number.
- Fare basis.
- Segment board and off point, since this thesis only adresses one segment (AAA-BBB).

3.1.1 Data Cleaning

This section contains some examples of the cleaning and transforming that is done. On purpose, some actions are left out since there would have been too many specific cases.

Instead of having 200 different countries in Point of sale for instance, the countries are grouped into fewer levels which later on is fed to the models.

$$PoS = PoS_{SE} + PoS_{DK} + PoS_{NO} + PoS_{other}$$

Data that for instance is duplicated, outliers such as flights booked too long in advance, the return flight is scheduled before the outbound flight were removed. For some reason the dates 11-16 in June 2016 is overrepresented and the flights in this time period also have a slightly higher no-show rate.

The data set also contains large amounts of missing data (NA's). In some cases, the NA's are removed and in some cases they need to be imputed. The imputation is different for each covariate but usually done by a clever grouping and taking the average of the covariates not containing NA's.

The covariate 'travel purpose' is an estimation since when performing online booking, customers usually fail to specify whether they are travelling with a business or leisure purpose. The decision ruling for how to estimate the travel purpose is something not discussed here, but a quite simple model is used. This is something that is fairly complex and people in the airline industry are working on complex models trying to estimate and/or compute the travel purpose of passengers.

The data is also grouped by long and short haul, i.e. the length of the flight since different no-show behaviour for the two groups is expected. This is done by calculating the distance between the airports for each segment.

Some booking classes could be translated into a level of refundability which is believed to have a high impact on how likely the passengers are to be no-shows. So for instance, booking classes C,D and Z are always fully refundable since they correspond to first class tickets.

3.2 Baseline Model

In order to implement machine learning algorithms to try to predict the no-show rate given the PNRs a baseline model is needed for comparison. If machine learning can't beat this simple benchmark model there is probably no point using machine learning when predicting no-show rates. The model that Amadeus uses today for no-show forecasting is fairly simple but with many specific groupings. The model takes the historical no-show rate for each flight on a given segment, departure time, day of week, season and so on, but still a historical average. The original baseline model is kept secret but a similar model to the baseline is implemented by hand and this by hand implemented model is used as a benchmark.

Our by hand implemented baseline model groups on a segment level, the departure time (with levels Morning/Day/Evening), day of week and cabin. Imagine one is looking at the segment AAA-BBB. Then for 9 months of data, each unique flight is obtained, the no-show rate calculated and then the average no-show rate given the time of day, cabin and day of week is computed.

Table 3.1: Baseline model. AAA-BBB, cabin: M

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Morning	0.0213	...					
Afternoon	...	...					
Evening							...

Since we are only looking at 2 different cabins, M and C, the output from the baseline model is for each segment, 2 matrices containing the historical average of the no-show rates for that segment and cabin. Diving into 2 cabins is good for a first analysis but the airlines usually uses some sort of cross cabin optimization when reserving seats in a flight. An example would be a flight with a 100 seats and 20 intended business class seats. If one would sell more than 20 business class seats, the steward would simply move the curtain further back, hence, making a larger C cabin. This would lead to the opportunity to sell more low cost tickets to still keep the economic balance on this specific flight. Therefore, doing no-show forecasting on different cabins makes sense.

After obtaining these matrices, each unique flight and its no-show rate in the test months are then obtained and an error matrix is created as the actual no-show rate minus the predicted. Where the prediction from the two matrices given by the baseline model is obtained, depends on when the flight in the test month is and what day of week it is. Since the grouping is done by cabin, there will be two error matrices, one for each cabin.

$$e = act_{NSR} - pred_{NSR} \tag{3.1}$$

The error matrices are simply the actual (*act*) no-show rates minus the predicted (*pred*) no-show rates.

3.3 CART - Recursive Partitioning

In R, a library called *rpart*(), meaning recursive partitioning can be used to create decision trees. The packages are installed and called via the following lines of code.

```
library(rpart)
library(rpart.plot)
```

Training a model using rpart, given the covariates and settings.

```
tree <- rpart(NSR~., data = train, cp = -1, control = list(maxdepth = 25)),
```

where *tree* is the name of the model, NSR the no-show rate we want the model to have as output, the $data = train$ is the training set, *cp* the complexity parameter denoted as α in equation (2.14) and the -1 meaning that the parameter can take on all values without being bounded. The last parameter, *maxdepth* meaning how many splits the tree is maximum allowed to have.

When training the tree, rpart by default does a 10-fold cross validation on the given data set (in this case the training set) in order to find the optimal complexity parameter α. After doing this for a large number of different complexity parameters, the complexity parameter can be plotted versus the relative error. Below is a plot of the complexity parameter against the relative error.

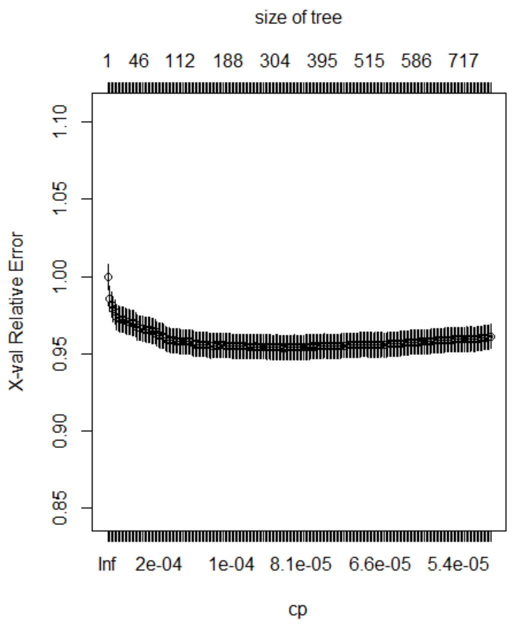


Figure 3.1: Complexity parameter against the relative error. The size of the tree can also be seen on top of the figure.

The tree is then pruned with the complexity parameter that minimizes the relative error by using the following code

```
pruned = prune(tree,cp = tree$cptable[which.min(tree$cptable[,"xerror"]),"CP"])
```

Where *pruned* is the pruned model of *tree*. The model *pruned* is then used for prediction.

Predicting using a model is done by using the built in function *predict* in R.

```
pred = predict(pruned, test, type = "vector")
```

This means that the output *pred* is a vector of, in this case, probabilities of no-show per PNR, obtained from the model *pruned* when the model is given the test data (*test*).

3.4 Extreme Gradient Boosting

A famous version of the gradient boosting algorithm, called extreme gradient boosting with shortening *XGboost* is used in R [24]. It's called *xgboost*() just as the shortening and XGboost is a well known since it's performed well on real data sets and won competitions, for instance on *Kaggle.com*.

The libraries useful when using *xgboost*() were obtained via the following lines of code.

```
library(xgboost)
library(caret)
library(Matrix)
library(Ckmeans.1d.dp)
library(DiagrammeR)
library(xtable)
```

The *xgboost*() algorithm in R need to have sparse format data as inputs. Sparse matrices are matrices that mostly contain 0. Therefore, the input matrix needs to be dummy coded with a function called *sparse.model.matrix*() in R. The train and the test set are then created.

```
sparse_train <- sparse.model.matrix(NSh~.-1, data = train[,clm])
sparse_test <- sparse.model.matrix(NSh~.-1, data = test[,clm])
```

The *xgboost*() algorithm in R needs the label of the output in order to form a *xgb.matrix* object. These are formed and added to a matrix format optimized for the *xgboost*() algorithm.

```
    xgb_train <- xgb.DMatrix(data = as.matrix(sparse_train), label=train[,'NSh'])
xgb_test <- xgb.DMatrix(data = as.matrix(sparse_test), label=test[,'NSh'])
```

Training an *xgboost*() model can be done by using the following code:

```
    xgb1 <- xgboost(data = xgb_train, booster = 'gbtree', max_depth = t,
    eta = e, nrounds = n, min_child_weight = m
 objective = "binary:logistic", eval_metric = 'auc')
```

the training set is *xgb_train*, the method used is gradient boosting tree ('gbtree'),*max_depth* is how many splits each decision tree in each round is maximum allowed to have. The *'eta'* is the learning rate and the objective function is to predict a binary logistic output i.e an output $\in [0, 1]$. The evaluation metric is *'auc'* or area under the curve which briefly explained is a measurement of how accurate a binary classifier is. An $auc = 0.5$ is completely random and an $auc = 1$ is it predicts perfectly and *auc* is closely related to false/true positives/negatives.

In order to find the "best model", cross validation is used to find optimal tuning parameters for the *xgboost*() model. A grid is created containing different *max_depth*, learning rates *eta* and *min_child_weight* meaning, in this case, the minimum number of examples in a node to make a new split. For each combination of these tuning parameters, the model is cross validated and the best performing combination, according to the *auc*, of the tuning parameters is then used when training the model used for prediction.

When the optimal tuning parameters are found, the model is then trained ”normally” on the training set since the best parameters via cross validation for this particular data set are found.

Predict using *xgboost*() in R is smoothly done by using R's built in function *predict*.

```
    pred <- predict(xgb1,xgb_test)
```

This means that the output *pred* is a vector of, in this case, probabilities of no-show per PNR, obtained from the model *xgb*1 when the model is given the test data *xgb_test*.

3.5 Neural Network

In R, there are several libraries that work with neural networks. In this case, the library called *neuralnet*() is chosen for its simplicity and easy to use multilayer perceptron implementation. This library only has the ability of having a single output node in the output layer. This is fine, since we are doing binary classification/regression meaning that obtaining the probability of being a no-show is enough as an output value. Instead, if a regression is made with an output with more than two levels, then multiple output nodes is needed and the specific library *neuralnet*() can't be used. In that case, libraries like *nnet*() are more suitable.

The libraries useful when working with *neuralnet*() are obtained by the following lines of code.

```
library(caret)
library(neuralnet)
```

The *neuralnet*() algorithm in R needs to have a special format on the inputs which can be created using the *caret*() package. It is a version of dummy coding which is commonly used in R. One way of dummy coding the train and the test set is by using the following lines of code.

```
dummytrain <- dummyVars("~.", data = train1)
dummytrain <- data.frame(predict(dummytrain, newdata = train1))
dummytest <- dummyVars("~.", data = test1)
dummytest <- data.frame(predict(dummytest, newdata = test1))
```

This is done on the scaled inputs. The inputs are either standardized using normalization (see section 2.2.2) or if the covariates only had two levels, transformed into binary variables with levels $[-1, 1]$.

Training the network can be done by running the following code.

```
nn <- neuralnet(f, data = dummytrain, hidden = layers,
err.fct = "ce", linear.output = FALSE,
lifesign.step = 100, lifesign = "full", threshold = t,
learningrate.factor = list(minus = a, plus = b)),
```

where f is the formula being used, i.e what should be predicted using what. The command *hidden* is the number of hidden layers and how many nodes in each layer that should be used. The command $err.fct$ is the error function which in this case is chosen to be the cross-entropy function as mentioned earlier in equation (2.11). The command *linear.output* means that the activation function should be applied to the output node.
The lifesign command just states what the algorithm should show and its progress when training. The *threshold* is an early stopping criteria and the $learningrate.factor$ is the range of which values the learning rate is allowed to have.

Predicting using the *neuralnet* package is pretty straight forward. Instead of using R's built in function *predict*, the function *compute* is used. Compute returns both the neurons in the trained network and the values in the output layer. The following lines of code returns the output layer for the test set.

```
 nhat_net <- compute(nn, dummytest)
pred_net <- unlist(nhat_net[2])
```

Chapter 4

Results

The first section in this chapter shows plots obtained from Tableau, a software used for graphic illustrations of data, intended to give a basic understanding of the data set.

The other sections, one for each machine learning algorithm, the density of the prediction error is presented. The prediction error is measured in no-show rate, a 0.1 prediction error could for instance mean that the actual no-show rate is 0.2 but the predicted one was 0.1. The densities are presented as plots with one plot for each cabin and model used for predicting.

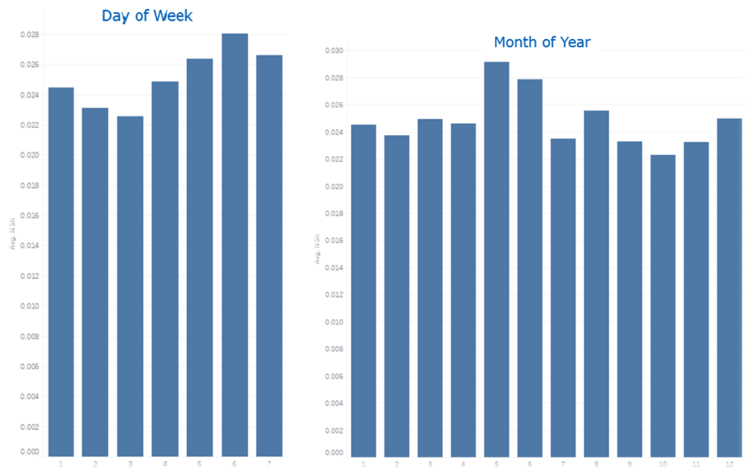
Then the actual no-show rates are plotted vs the predicted ones. This is also presented as plots with one for each cabin and model used for predicting.

Lastly, a table is presented showing the KPI's explained in section 2.4.3, where each model is compared to the baseline model on a cabin level. The only KPI not mentioned in section 2.4.3 is the one called SD which is the standard deviation of the error. The best result for each category, out of all the models (including the baseline), is highlighted in orange. Hence, some tables are containing less orange colours than others.

All the results are from the same flight route (AAA-BBB) and the predictions are made on the test month using the suitable data as explained before in chapter 3. This thesis only presents results from the segment AAA-BBB. Other tested segments (CCC-DDD) give similar results.

4.1 Data Exploration

These descriptive plots are plotted using the statistical software Tableau. They give a basic understanding of different group behaviour in the data set. Again, these plots describe the segment AAA-BBB and only a few are presented in this section.

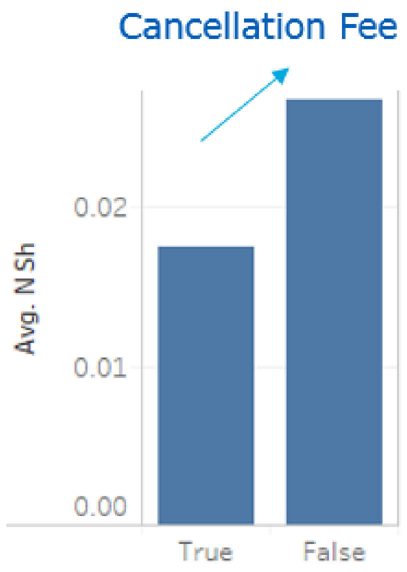


(a) No-show rate per Day of Week.

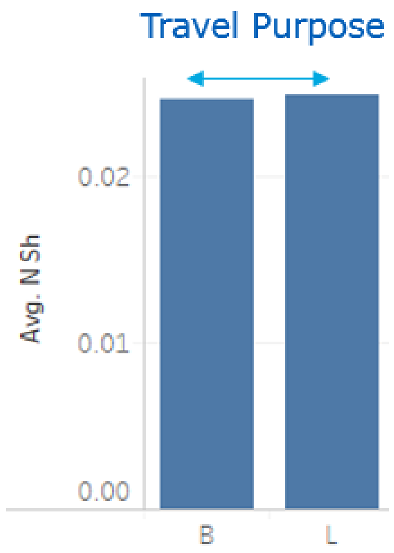
(b) No-show rate per Month of Year.

Figure 4.1: The no-show rates plotted per Day of Week and Month of Year during one year.

In the figures above, there is a very small difference in the no-show rates for Day of Week and Month of Year. The no-show rates per Day of Week lies within $[0.022, 0.028]$ and for Month of Year $[0.022, 0.029]$. Already here, finding a certain trend based on Day of Week or Month of Year seems difficult.



(a) No-show rate per Cancellation Fee.



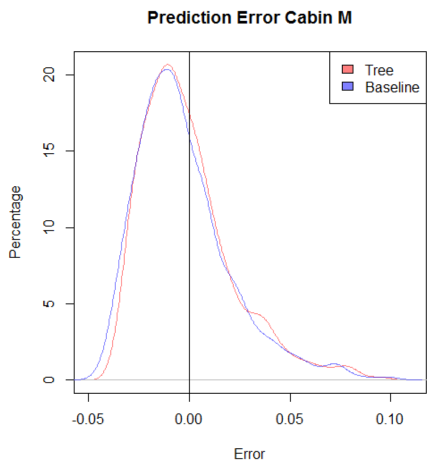
(b) No-show rate per Travel Purpose, B means business and L leisure.

Figure 4.2: The no-show rates plotted per Cancellation Fee and Travel Purpose during one year.

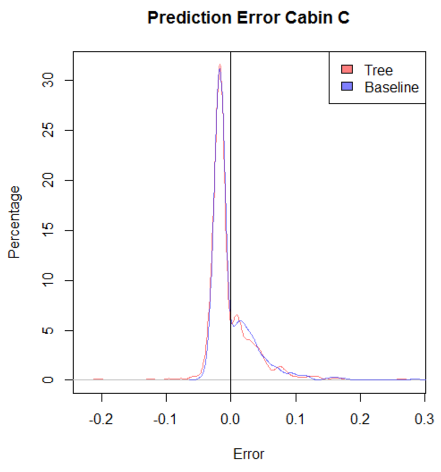
Cancellation Fee means if one would cancel his/her ticket, would it be fully refundable. If there is no cancellation fee (False) then the passenger would receive the full amount back for his/her ticket. When there is a cancellation fee (True), in this case, means that if the passenger cancels his/her ticket, they would receive some sort of fee which could vary from a few percent to the full ticket price.

In the figures above, there is a difference in no-show rates depending on if the passenger has a cancellation fee or not. A surprising result is that there is hardly any difference between the passengers with business and leisure travel purpose in their respective no-show rates. One would expect business passengers to have a higher no-show rate than leisure.

4.2 CART - Recursive Partitioning



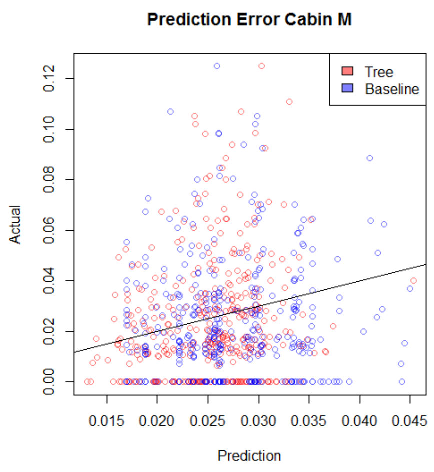
(a) Density, Cabin M.



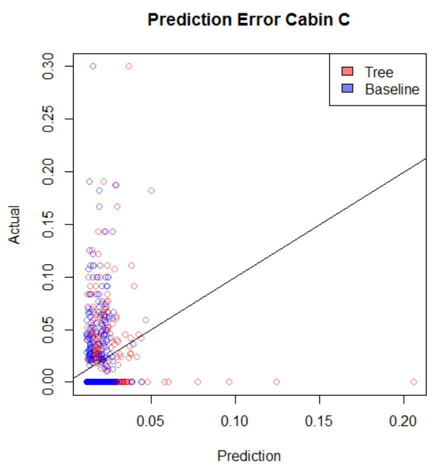
(b) Density, Cabin C.

Figure 4.3: Density of the prediction error for Cabin M and C. Predictions obtained via recursive partitioning tree and compared to the self implemented baseline model. The black line indicates where the error is 0.

In the figure above we see that the density for decision trees is very similar to the baseline model. Hard to argue if one is better than the other just by looking at these plots. One can see the bias, that the centering of the density is slightly shifted to the left for both cabins.



(a) Actual no-show rates vs predicted no-show rates, Cabin M.



(b) Actual no-show rates vs predicted no-show rates, Cabin C.

Figure 4.4: Actual no-show rates plotted vs the predicted no-show rates for Cabin M and C. Predictions obtained via recursive partitioning tree and compared to the self implemented baseline model. The black line indicates where actual no-show rates are equal to the predicted no-show rates.

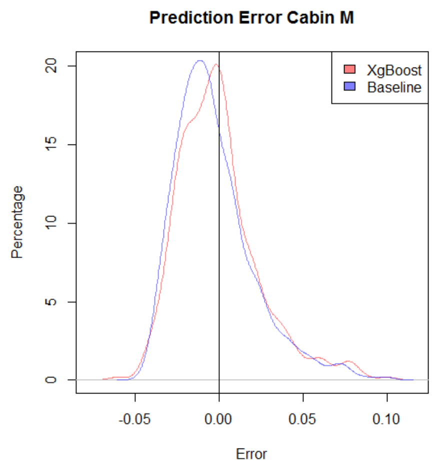
When looking at figure 4.4 above, a similar behaviour between the baseline model and the decision tree model is seen. Even the outliers are similiar. The concentration of predictions are higher than the actual no-show rates for the decision tree. This would lead to empty seats.

Table 4.1: KPI's for the baseline model and the CART-Recursive Partitioning Tree model for cabins M and C.

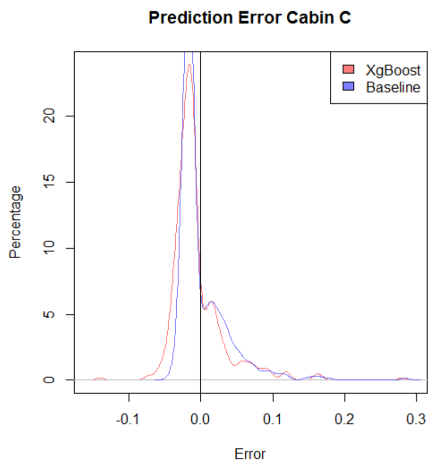
	ME	RMSE	MAE	MASE	SD	Accuracy	Precision	Recall
Baseline M	-0.0027791660	0.02376682	0.01859967	1.010469	0.02363444	0.9760211	NA	0
Baseline C	0.0010025106	0.03635059	0.02508650	1.008040	0.03638731	0.9832386	NA	0
Tree M	-0.0008526821	0.02310334	0.01773797	0.986756	0.02311805	0.9759307	0	0
Tree C	-0.0023875786	0.03804951	0.02634056	1.011136	0.03802597	0.9829424	0.4137931	0.04240283

Comparing table 4.1 with the baseline model, the models perform fairly similar but one could argue that the tree model performs slightly better. As can be seen, the tree model has the best results out of all the models in RMSE, MAE, MASE, SD, precision and recall. One important thing to notice is that the difference is not that large. Here, one can truly see the effect of a skewed class. The accuracy is really high for both the baseline and the tree model. However, the precision and recall results are really bad. So even if the Tree model performs best in precision and recall, the overall results in the KPIs precision/recall are pretty bad.

4.3 Extreme Gradient Boosting



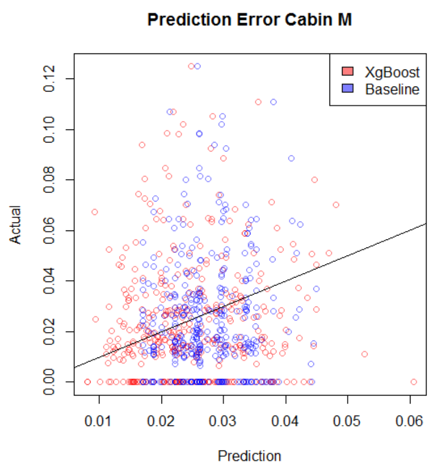
(a) Density, Cabin M.



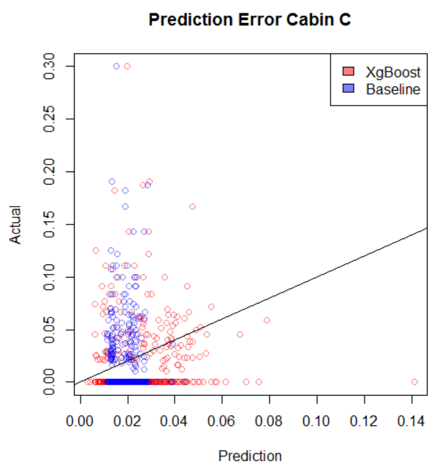
(b) Density, Cabin C.

Figure 4.5: Density of the prediction error for Cabin M and C. Predictions obtained via extreme gradient boosting and compared to the self implemented baseline model. The black line indicates where the error is 0.

When looking at figure 4.3 above, the density is fairly similar to the baseline model, and one might see that the density for the gradient boosting model is slightly more centered around 0, which is good. This is perhaps more visible in the left plot showing the density for the M cabin.



(a) Actual no-show rates vs predicted no-show rates, Cabin M.



(b) Actual no-show rates vs predicted no-show rates, Cabin C.

Figure 4.6: Actual no-show rates plotted vs the predicted no-show rates for Cabin M and C. Predictions obtained via extreme gradient boosting and compared to the self implemented baseline model. The black line indicates where actual no-show rates are equal to the predicted no-show rates.

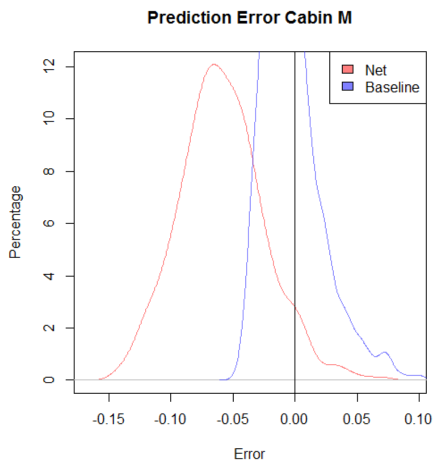
When looking at the figure above, it seems like the *xgboost* model has a larger spread and that the baseline model is more concentrated. It seems like the *xgboost* model has a higher tendency to predict higher no-show rates than the baseline model, at least in cabin C.

Table 4.2: KPI's for the baseline model and the extreme gradient boosting model for cabins M and C.

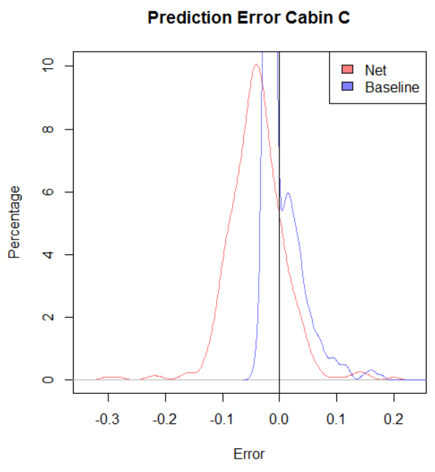
	ME	RMSE	MAE	MASE	SD	Accuracy	Precision	Recall
Baseline M	-0.0027791660	0.02376682	0.01859967	1.0104688	0.02363444	0.9760211	NA	0
Baseline C	0.0010025106	0.03635059	0.02508650	1.0080404	0.03638731	0.9832386	NA	0
XGB M	-0.0004475768	0.02434071	0.01815134	1.0138778	0.02436870	0.9758764	0	0
XGB C	-0.0054834911	0.03837400	0.02714368	0.9939287	0.03803096	0.9830609	0	0

When looking at the KPIs for the gradient boosting model we see that the mean error (ME) is the best out of all the models, meaning that it has the least bias. In the other KPIs it performs slightly worse than the tree model but generally not by a lot.

4.4 Neural Network



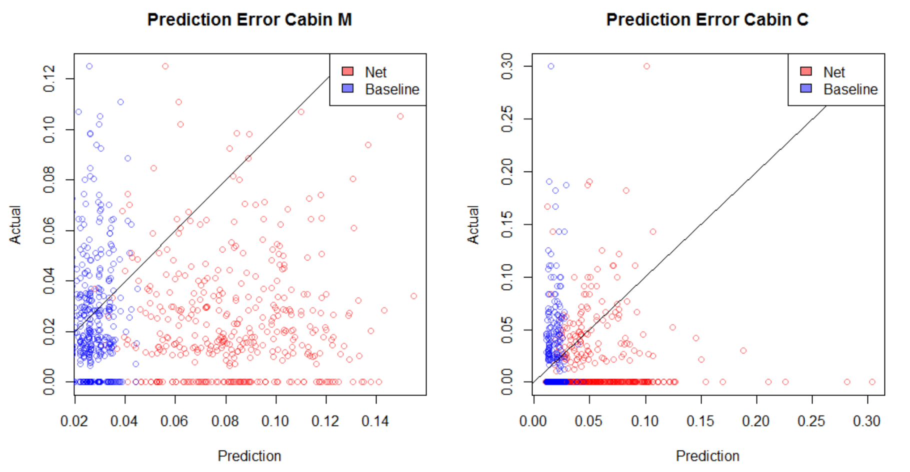
(a) Density, Cabin M.



(b) Density, Cabin C.

Figure 4.7: Density of the prediction error for Cabin M and C. Predictions obtained via Neural Network and compared to the self implemented baseline model. The black line indicates where the error is 0.

It's quite easy to see that for this particular neural network, the baseline model is much better when looking at the densities. The neural network both has a centering further to the left (more bias) and the density is also wider indicating larger standard deviation.



(a) Actual no-show rates vs predicted no-show rates, Cabin M.

(b) Actual no-show rates vs predicted no-show rates, Cabin C.

Figure 4.8: Actual no-show rates plotted vs the predicted no-show rates for Cabin M and C. Predictions obtained via Neural Network and compared to the self implemented baseline model. The black line indicates where actual no-show rates are equal to the predicted no-show rates.

In these figures we can easily tell that the neural network generally predicts a higher no-show rate than it actually is. It is quite clear, at least for the M cabin that this is the case. This would lead to denied boardings which in general is more expensive than having empty seats on a flight.

Table 4.3: KPI's for the baseline model and the Neural Network model for cabins M and C.

	ME	RMSE	MAE	MASE	SD	Accuracy	Precision	Recall
Baseline M	-0.002779165978	0.02376682293	0.01859966647	1.010468814	0.02363444100	0.9760211376	NA	0
Baseline C	0.001002510570	0.03635058610	0.02508650418	1.008040409	0.03638731113	0.9832385691	NA	0
Net M	0.058919990787	0.06802120785	0.06081758104	1.015094689	0.03403310176	0.9752791502	0.044	0.0015094
Net C	0.039840877558	0.06510740103	0.05145378972	1.052087580	0.05156585807	0.9809879176	0	0

As expected, when having looked at the densities and the actual vs predicted plots, the KPIs also indicate that this neural network has performed worse than the baseline, tree and gradient boosting model.

Chapter 5

Discussion

In this section, some of the main problems are discussed, the results are commented on and a few thoughts on the future are mentioned.

As glamourous as it might sound working with machine learning most of the work lies in the data set. Quoting my supervisor Olivia Mala: ”Machine learning is not about blindly plugging in data into black box models, it’s mostly about understanding your data set.” She couldn’t be more right.

The data set has throughout the process been changed regularly, sometimes due to bugs and sometimes due to more practical things. When obtaining a data set in a University course, everything just works, it’s been constructed so that there is some correlation, no missing data and the meaning of each covariate makes sense. This is not the case in the real world. Firstly, understanding all the airline terminology and what can be used as what took quite some time and I still feel that not everything is completely clear to me. Cleaning, changing, transforming and understanding the data set is where I’ve spent most of my time. As of today, there are still covariates that are excluded which later on could be added and that might yield better forecasting.

After deciding upon which segment and data set to use for modelling, the problem of skewed classes really emerged. After running it a few times and almost everytime achieving above 97% accuracy on the test set was too good to be true. First, I thought that the model was overfitting, but that didn’t make any sense since then the accuracy would have been really high on the training set and low on the test set which was not the case. Again, looking at accuracy was a misleading approach since all models performed really well in this KPI.

I spent quite some time trying to deal with skewed classes by penalising false positives, trying to regularize certain parameters and stratified sampling. The stratified sampling was quite interesting. The idea is that you feed a training set to the models which contains a different distribution of the classes.
For instance, I tried feeding the models a stratified sampled training set, a training set which contained 100000 PNRs, where half of them were no-show PNRs. The idea behind it is for the model to really learn to distinguish between a show and a no-show PNR. This worked, in the sense that the models got higher scores in precision and recall but lower accuracy.
However, the improvements in precision and recall weren't good enough. The models might have been better at specifically classifying PNRs as show/no-shows but on average they performed worse when predicting the no-show rates. Since overbooking uses the no-show rate of a flight as an input directly and the goal was to improve the quality of the forecasting of this input, then stratified sampling performed worse in this sense. Nevertheless, stratified sampling is something that could be interesting to look into further in the future.

The models used in the project are fairly different from each other and have several advantages/disadvantages. According to the results, the decision tree model performed the best in all but two KPIs. Decision trees are the simplest model out of the 3 different machine learning algorithms which can be seen as an advantage. It is easy to train, fast and easy to understand. It is far from a black box model, the tree can be plotted and one could see exactly why a decision has been made.
I've chosen not to plot the tree since I have so many covariates and fitting it into an $A4$ paper is hard, but upon request, it could be presented on a computer. This is definitely an advantage if this model were chosen for production, since customers can see themselves what the model does which isn't really possible when talking about for instance neural networks. A further investigation could be to test the machine learning algorithm Random Forest which is an ensemble method that takes many small decision trees where each tree tries to model the no-show rates using just a few covariates. These are then weighted together to create a more robust model. Random Forest is in many practical areas rather used than the decision tree algorithm.

The algorithm gradient boosting were just outperformed by the decision tree algorithm according to the chosen KPIs. This doesn't necessarily mean that this algorithm is worse. This remains a mystery why, since a gradient boosting model is a tree model that enhances itself. Worth mentioning is that the extreme gradient boosting algorithm requires careful tuning and it's likely that the model could be tuned better. Someone with more experience working with this algorithm might get better results and therefore this algorithm should not be overlooked. It is known to perform really well on real world data and is one of the algorithms that win most competitions on *kaggle.com*.

So, should the neural networks be discarded in this case? They did perform way worse than the other models. Well, they are more advanced and they are a black box model which might be harder to sell. If they do something wrong, there is no ”straight fix”. I found that the network I used had trouble converging and therefore the results might not be as good as the other models, I had to make them stop early. They are also known to be hard to train but I'm sure that with the right training they would also perform as well as the other algorithms, I just didn't have the time to cross-validate all the different types of learning rates, number of hidden layers and nodes etc.

Another important thing to add here is that all the results and analyses are based on a single segment AAA-BBB for a specific airline. Other segments we know for sure have a different behaviour than the chosen one. There is also a difference if the chosen segment typically is a business or a leisure segment or a long or a short haul segment etc. We also know that different airlines have different no-show behaviour and different policies regarding this. Since these models haven't been tested for other airlines in this thesis, it's hard to give a general decision whether machine learning should be used when forecasting no-show rates. One thing that the project show, based on the obtained results, is that machine learning at least doesn't make the forecasting worse (when looking at decision trees and gradient boosting).

Important to mention, PNR-based forecasting is something that airlines have requested and is not just a research area, it is being used today.

Regarding the business side of the project, is it better to have a model that more often predicts a too high or low no-show rate? As mentioned before, it is generally speaking cheaper to have an empty seat than a denied boarding when looking at a single passenger. But when taking into account that never having a denied boarding means that you always overbook too little and that the airline isn't taking enough risk. So you can see it that overbooking more aggressively means that the airline is taking more risk and therefore, more often than not, will have a higher expected return.
Of course, there are cases when denied boarding is more unpleasant than other times. For instance, if you are booked on a flight from Vilhelmina going to Stockholm and the next flight leaves in 4 days, then a denied boarding is much worse than if you are flying from Stockholm to London and you are denied boarding but the next flight leaves in an hour. Therefore choosing a model that estimates the no-show rates according to your business requirements of importance if one needs to choose. Of course, the best is to have a model that simply forecasts the no-show rates perfectly, that goes without saying.

Apart from the other things mentioned before about the future there are some interesting models that could be looked into. Banks use something called anomaly detection when detecting for instance credit card fraud. These are very rare events since most transactions are not fraud and can be compared to a very skewed class. There are also more statistically based models called zero-inflated models that also are based on a distribution with a majority of a 0 class.

A final thought is of course to go further into AI and try to use reinforcement learning. Reinforcement learning is the type of AI that Google Deepmind used to beat the Chinese board game Alpha-go. The algorithms works with rewards and the cost function is then defined as a sum of rewards where the AI gets penalised when it makes an error and rewarded when it takes a correct action. So a simple idea could be to reward the AI for correctly classifying a no-show passenger with a relative high reward and penalise it relatively heavy when it makes a wrong no-show prediction. I find reinforcement learning to be the far most interesting area within AI but also probably the most advanced as well. All these models could be looked into in the future.

Chapter 6

Conclusions

We have considered the problem of using machine learning algorithms in order to improve forecasting the no-show rate of passengers. The goal was to see if the simple baseline model could be beaten by decision trees, gradient boosting and/or neural networks and as of now, the answer is not clear. Decision trees scored the best in our chosen KPIs (Key Performance Indicators) and gradient boosting had similar but slightly worse results. The difference is so small that one could argue if there even is a difference between the baseline, decision trees and gradient boosting. The neural network performed the worst out of all the models and should rather be more looked into as a second approach.

The most likely reason why the algorithms didn't perform very well is due to the randomness of the problem, a passenger just doesn't show up sometimes. It might have been intuitive that having more information about the passenger would help. However, the methods and the knowledge obtained can still be used. It seems promising when moving on to cancellations, which have a much higher rate and it might be more related to passenger behaviour.

This project has produced results showing that using machine learning when forecasting no-show rates using the PNRs is possible. I still feel that there are many areas to be explored and some might work better than others. If I today had to choose one of the methods in the thesis for production, I would choose decision trees as a first step. As a second step I would probably expand to using Random Forest [9], and I'm saying this without having tested the algorithm since in many peoples opinion it is a more robust "extension" of the decision tree algorithm.

Appendix A

Covariates with brief explanation:

- Origin board point.
- Origin off point.
- Departure date.
- Departure day of week.
- List of Airports - All the airport the Origin/Destination contains.
- Number of segments - A segment is each ”stop”. E.g Arlanda-New York, where Arlanda-Frankfurt, Frankfurt-New York are 2 different segments.
- Booking number.
- Days to departure - number of days before departure the reservation was made.
- Creation day of week.
- Creation time.
- Number of nights - how many nights are the passengers away.
- Day details - which days are they away.
- Travel Direction - One way/Outbound/Return.
- Yield - what the ticket was calculated to cost.
- Fare - what the passenger actually paid.
- Fare basis - code which translates to different rules regarding the booking.
- Company code - is the ticket booked by privately or by a company.
- Is group - in the airline industry, a group is more than 10 people in the same booking.
- Point of sale - which country was the booking made in.
- Is ticketed - Is the ticket issued or not.

- Is staff - If the booking is by airline staff.
- Is Crew - If the booking is by airline crew.
- Segment ID - which number is the segment.
- Segment board point.
- Segment off point.
- Segment departure date.
- Segment departure time.
- Segment departure day of week.
- Connection time.
- Cabin - A grouping in the airline industry.
- Booking class.
- Pax type - what type of passenger is it, Adult, Child or Infant.
- Status - Boarded/no-show.
- Travel purpose - Business/Leisure.
- Number in party - how many people in the same booking.
- Cancellation fee

Appendix B

First column corresponds to the cases IDs, only 6 rows are shown in this example. Again, emphasizing that this data is fabricated, the numbers and letters are made up by me. The data in Appendix *B* is fabricated due to company secrecy.

```
   X OBP OOP    DepDate DepDoW DepTime AirportList NofSeg Recloc DtD
1  7 AAA BBB 2016-01-02      1     700 AAA-CCC-BBB      1 YABP7X  2
2  8 AAA BBB 2016-02-03      2     800 AAA-CCC-BBB      2 YABP7X  2
3  9 ABC DEF 2016-03-04      3     900 ABC-CCC-DEF      3 YAPP7X  26
4 10 ABC DEF 2016-04-05      4     1000 ABC-GHI-DEF     1 YAPP7X  26
5 11 ABC DEF 2016-05-06      5     1100 ABC-GHI-DEF     2 YAPP7X  26
6 12 ABC DEF 2016-06-07      6     1200 ABC-GHI-DEF     3 YAPP7X  26

 CreationDoW CreationTime NofNights DayDetails Direction Yield   Fare     FareBasis CompCode A
1           6          2216         4       2345        OW  1000   900      USED-RTM1      1    4
2           6          2216         4       2345        OW  1000   900      USED-RTM2      1    4
3           6          2316         5       12345       RT  2000  1219.84   USER-RTM3      0    2
4           6          2316         5       12345       RT  2000  1219.84   USER-RTM4      0    2
5           6          1210         5       12345       RT  2000  1219.84   USER-RTM5      0    2
6           6          1210         5       12345       RT  2000  1219.84   USER-RTM6      0    2

  C I PoS IsTicketed IsStaff IsCrew SegID SBP SOP SDepDate
1 2 0  GB          1       0      0     2 CCC BBB   280116
2 2 0  GB          1       0      0     2 CCC BBB   280216
3 2 0  GB          1       0      0     2 CCC DEF   280316
4 2 0  GB          1       0      0     2 GHI DEF   280416
5 2 0  GB          1       0      0     2 GHI DEF   280516
6 2 0  GB          1       0      0     2 GHI DEF   280616

 SDepTime CnxTime Cabin BC CnlDtD PaxType Status TP Yield_comp Fare_comp
1       920      85     M  U     NA       A     BD  L       1000    900.84
2       920      85     M  U     NA       A     BD  L       1000    900.84
3       920      85     M  U     NA       A     BD  L       2000    1200.84
4       920      85     M  U     NA       A     BD  L       2000    1200.84
5       920      85     M  U     NA       C     BD  L       2000    1200.84
6       920      85     M  U     NA       C     BD  L       2000    1200.84
```

```
 NiP CnlFee OnDType     OnDDist     SegDist PoS_comp Day1 Day2 Day3 Day4
1   6  FALSE  EurBus 700.4871971 334.9607789      RoW    1    1    1    1
2   6  FALSE  EurBus 700.4871971 334.9607789      RoW    1    1    1    1
3   6  FALSE  EurBus 911.4871971 354.9607789      RoW    1    1    1    1
4   6  FALSE  EurBus 911.4871971 354.9607789      RoW    1    1    1    1
5   6  FALSE  EurBus 911.4871971 354.9607789      RoW    1    1    1    1
6   6  FALSE  EurBus 911.4871971 354.9607789      RoW    1    1    1    1

 Day5 Day6 Day7 NofDays     ToD    SToD SDepYear SDepMonth NSh  SDepDoW
1    1    0    0       5 Morning Morning       16         1   0       1
2    1    0    0       5 Morning Morning       16         2   0       2
3    1    0    0       5 Morning Morning       16         3   0       3
4    1    0    0       5 Morning Morning       16         4   0       4
5    1    0    0       5 Morning Morning       16         5   1       5
6    1    0    0       5 Morning Morning       16         6   0       6
```

Bibliography

[1] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. "A study of the behavior of several methods for balancing machine learning training data". In: *ACM Sigkdd Explorations Newsletter* 6.1 (2004), pp. 20–29.

[2] Dimitri P Bertsekas and Sanjoy K Mitter. "A descent numerical method for optimization problems with nondifferentiable cost functionals". In: *SIAM Journal on Control* 11.4 (1973), pp. 637–652.

[3] Adrian W Bowman. "An alternative method of cross-validation for the smoothing of density estimates". In: *Biometrika* 71.2 (1984), pp. 353–360.

[4] Michael Buckland and Fredric Gey. "The relationship between recall and precision". In: *Journal of the American society for information science* 45.1 (1994), pp. 12–19.

[5] Adam Coates and Andrew Y Ng. "The importance of encoding versus training with sparse coding and vector quantization". In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 921–928.

[6] Devexploria. *Multilayer percepteron with bias term.* [Online; accessed November 16, 2016]. 2012. URL: `http://retomatter.blogspot.se/2012/12/functional-feed-forward-neural-networks.html`.

[7] Pedro Domingos. "A few useful things to know about machine learning". In: *Communications of the ACM* 55.10 (2012), pp. 78–87.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* Book in preparation for MIT Press. 2016. Chap. 5,6,12.

[9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[10] Moshe Leshno. "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function". In: *Neural Networks* 6.6 (1993), pp. 861–867.

[11] Richard Lippmann. "An introduction to computing with neural nets". In: *IEEE Assp magazine* 4.2 (1987), pp. 4–22.

[12] Michael A. Nielsen. *Neural Networks and Deep Learning.* Determination press, 2015. Chap. 3.

[13] Joseph O Ogutu, Hans-Peter Piepho, and Torben Schulz-Streeck. "A comparison of random forests, boosting and support vector machines for genomic selection". In: *BMC proceedings*. Vol. 5. 3. BioMed Central. 2011, p. 1.

[14] Robert L. Philipps. *Pricing and Revenue Optimization*. Stanford, CA, USA: Stanford University Press, 2005, pp. 207–238.

[15] Foster Provost. "Machine learning from imbalanced data sets 101". In: *Proceedings of the AAAI'2000 workshop on imbalanced data sets*. 2000, pp. 1–3.

[16] Dorian Pyle. *Data preparation for data mining*. Vol. 1. Morgan Kaufmann, 1999, pp. 224–252.

[17] Marvin Rothstein. "OR and the Airline Overbooking Problem". In: *Operations Research* 33.2 (1985), pp. 237–249.

[18] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *BMC proceedings*. Insight Centre for Data Analytics, NUI Galway Aylien Ltd., Dublin. 2016, pp. 1–3, 7.

[19] Joseph L Schafer. "Multiple imputation: a primer". In: *Statistical methods in medical research* 8.1 (1999), pp. 3–15.

[20] P.E Pfeifer S.E Bodily. "Overbooking Decision Rules". In: *OMEGA Int. J. of Mgmt Sci.* 20 (1991), pp. 129–133.

[21] Stackexchange. *Over and underfitting - figure*. [Online; accessed November 27, 2016]. 2014. URL: `http://datascience.stackexchange.com/questions/361/when-is-a-model-underfitted`.

[22] Wil MP Van der Aalst et al. "Process mining: a two-step approach to balance between underfitting and overfitting". In: *Software & Systems Modeling* 9.1 (2010), pp. 87–111.

[23] Vladimir Vapnik. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer, 1999, pp. 18–23. ISBN: 978-1-4419-3160-3.

[24] *XGBoost R Tutorial*. `https://xgboost.readthedocs.io/en/latest/R-package/xgboostPresentation.html`. Accessed: 2017-03-02.