Master Thesis Business Analytics

Detecting the Arrivals and Departures of Vehicles with Machine Learning Techniques

Marije Gemmink

July 2021

Graduation Supervisor:
Guszti Eiben

Second Reader:
Rob van der Mei

External Supervisor:
Martijn Loot
Silvie Spreeuwenberg

Vrije Universiteit Amsterdam
Faculty of Science
Business Analytics
De Boelelaan 1081a
1081 HV Amsterdam

Simacan
Valutaboulevard 16
3825 BT Amersfoort

Preface

This thesis is the result of an internship of the Master Business Analytics. The internship is the final course of the two-year program at the Vrije Universiteit Amsterdam. The internship lasted six months and was conducted at the Data Research department at Simacan in Amersfoort. Simacan supports logistics operations by receiving and processing planning, monitoring the execution of the logistics operation by receiving and processing data from the driving vehicles. All information is brought together to arrive at an accurate overview of insight and to make it easy for parties in and between companies to collaborate and communicate with each other.

This research focuses on developing a stop detection model of vehicles with machine learning models. It has been investigated whether machine learning can be used as stop detection and if this leads to better detections than the current Finite State Machine stop detection. The performance is better when more arrivals and departures are detected and more precise in time.

I would like to thank my supervisor at the VU, Guszti Eiben, and my external supervisors at Simacan, Martijn Loot and Silvie Spreeuwenberg, for their guidance. In addition, I would like to thank Martijn that despite leaving Simacan during my internship, he made time to provide feedback on my thesis. I am very grateful for the help and support during my internship period from all supervisors. I would also like to thank my colleague, Bob Soeters, for his time and feedback. In addition, I would like to thank my second reader, Rob van der Mei, for his time.

Finally, I would like to thank my parents, sister Anne and Derbert, for their loving support. They made me believe I could finish my Master's degree.

Executive Summary

Confidential

Contents

1 Introductions **4**

2 Literature Review **5**

2.1 Detecting Stop Locations 5

2.2 Predicting Arrival Times 6

3 Current Stop Detection Algorithm **7**

4 Methods **8**

4.1 Machine Learning Techniques 8

4.1.1 Supervised Machine Learning 8

4.1.2 Decision Tree 9

4.1.3 Random Forest 10

4.1.4 Gradient Boosting 11

4.1.5 XGBoost 13

4.1.6 Support Vector Machines 14

4.1.7 Artificial Neural Network 17

4.2 Performance Metrics 22

4.3 Cross Validation 23

4.4 Hyperparameter Tuning 24

4.4.1 Random Search and Grid Search 24

4.5 Summary 25

5 Data **26**

6 Results **27**

7 Conclusion **28**

Appendix A Transitions FSM Model **29**

Appendix B Parameters Arrivals **30**

Appendix C Detected Arrivals **32**

Appendix D Parameters Departures **33**

Appendix E Detected Departures **35**

Chapter 1

Introductions

Confidential

Chapter 2

Literature Review

This chapter provides an overview of previous research with Global Positioning System (GPS) related to this study. First, some literature about detecting the stop locations themselves is provided, followed by predicting the arrival time.

2.1 Detecting Stop Locations

The studies about detecting stop locations differ slightly from this study. The difference is that the studies below detect the stop locations themselves, while the stop locations in this study are already known.

Yang et al. (2014) implemented a supervised machine learning model, Support Vector Machine (SVM) with nested k-fold cross validation, for identifying urban freight delivery stops from second-by-second GPS data in New York City. This model is based on delivery stops with a minimum stop duration of around ten minutes. In total, 2249 stops were detected from the GPS data. Most of the identified stops were nondelivery stops, only 42 were delivery stops. The extracted features from the GPS data were: the duration of a stop, the distance from a stop to the center of New York City, and the distance to a stop closest to a major bottleneck. The ground truth labels, the delivery stops of the trip, were available due to the driver log data recorded by drivers of the delivery trucks. The accuracy of the linear SVM with nested 10-fold is higher than 99% for the training and testing data sets.

Another study to detect the locations of a stop was conducted by Ting et al. (2017). An unsupervised model was implemented, whereby a dynamic approach was used. Hereby no prior knowledge is given while stop locations can be discovered. A stop is a part of a trip where the vehicle has spent a minimal amount of time. The use of GPS trajectories discovers these stop locations. Each trajectory in the data set consists of temporal, ordered, timestamped points with geographical coordinate information such as longitude, latitude, and altitude. The basis of Ting et al. approach is that the stop of a trajectory should have a lower moveability and a higher density of GPS points. Therefore, finding an appropriate method to estimate the density when moveability is considered is meaningful. The original DBSCAN (Density-Based Spatial Clustering of Applications) method is improved to consider moveability by giving a lower moveability a larger weight. The improved DBSCAN model has an F1-score of 0.9583 against an F1-score of 0.7778 for the standard DBSCAN model. From this it follows that the improved model works better than the standard model.

2.2 Predicting Arrival Times

Research has also been conducted into predicting the arrival time at a stop. This is related to the subject of this study since the data consists of GPS points as well. However, these studies are not related to the research question. The difference with this research is that the studies below predict the arrival time, while we want to predict whether the vehicles have arrived or departed: yes or no. In this research, the detected arrival or departure time is derived from the timestamp of the corresponding GPS point.

Liu et al. (2020) collected a GPS data set (i.e., update frequency about 30 seconds) of buses' trajectories in Dublin and uses an efficient pre-processing method that compares widely used prediction models for predicting the bus arrival times. The pre-processing method contains two phases. Trip segmentation is performed first, followed by point interpolation. Point interpolation is performed in two ways: interpolating a GPS point every 100 meters and interpolating GPS points at a bus stop. These methods are called *distance-based trip* and *stop-based trip*, respectively. 4311 trips are selected after pre-processing the data, where each distance-based trip contains 191 points, while each stop-based trip includes 59 points. Predicting the arrival time is conducted on the two interpolation techniques and tested on five prediction models: delay, k-nearest-neighbor (k-NN), kernel regression (KR), additive model, and recurrent neural network using long short term memory (RNN-LSTM). RNN-LSTM performs best, followed by KR, k-NN, delay, and the additive model. Besides, the stop-based trip interpolation method outperforms the distance-based trip method, significantly reducing the training and prediction computational cost while maintaining satisfactory prediction accuracy.

A new bus arrival time prediction algorithm that combines GPS data with real-time estimates of inter-station travel speeds is developed by Sun et al. (2007). The two weeks' collected GPS data were logged with a fixed time interval of 26 seconds from a bus route in China. Before predicting the bus arrival time, the exact location of the bus on the route based on its GPS data is determined. After which, the average travel speed to the station is calculated, and the bus running direction. Finally, a FSM model is constructed with four states to predict the arrival times at a station. The four states are beginning, prediction, terminus, and unknown. The bus starts in the beginning state and transitions to the prediction state when the bus route and bus running direction are found. In the prediction state, the predicted arrival times to the stations continually update. When a bus reaches the final stop, it will transit to the terminus states. In every state, an unexpected incident can occur. In that case, transition to the unknown state takes place. The results indicate that the proposed system can achieve satisfactory accuracy in predicting bus arrival times and perfect performance in predicting travel direction.

As mentioned, the studies from this chapter are not entirely in line with this research since we want to predict if the vehicle has arrived at or departed from a stop location. In addition, we are dealing with a special situation too, namely unloading and loading at the same location, which is seen as two stops. Previous research on this topic has not yet been performed. Therefore different models will be implemented on different feature sets to investigate which combination gives the best performance. The used models and feature sets are discussed in Chapters 4 and 5, respectively.

Chapter 3

Current Stop Detection Algorithm

Confidential

Chapter 4

Methods

In the previous chapter, the operation of the current stop detection model was described. In this chapter it is investigated which machine learning models are best to apply for a stop detection model. First, a brief explanation about supervised learning is given, followed by a description of the different machine learning models which will be used in the following chapters of this research. Then, various performance metrics are explained. Performance metrics are needed to evaluate and compare the performance of the different models later in this research. Finally, a description of hyperparameter tuning is given, which is required to receive the best results.

4.1 Machine Learning Techniques

Machine learning is a field of artificial intelligence (AI). As Han et al. (2012a) describe, machine learning is a technique whereby computers can learn from data without any human knowledge. When using machine learning techniques, the computers will understand the problem complexity, recognize patterns and make intelligent decisions based on data. Machine learning algorithms can be divided into the following categories (Alzubi et al., 2018):

- Supervised learning: predict the outcomes for new data.
- Unsupervised learning: get insights from large volumes of new data.
- Semi-supervised learning: a combination of supervised and unsupervised learning.
- Reinforcement learning: during the learning process, a response tells whether the output is correct or not. The algorithm explorers and rules out various possibilities to get the correct output.

This research addresses supervised learning.

4.1.1 Supervised Machine Learning

According to Alzubi et al. (2018), Muhammad and Yan (2015), and Dey (2016), supervised machine learning algorithms are designed to learn by example. In addition to the independent variables, the training data also contains the correct answer, the truth labels. During training, the algorithm will search for patterns in the data that correlate with the desired output. After training, the algorithm will take new unseen inputs from the test set and determine which label the new inputs will be classified based on prior training data. The objective is to predict the correct label for the test data. A visualization of the supervised learning process is in Figure 4.1.

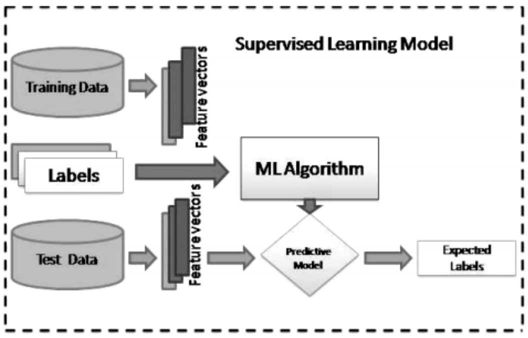


Figure 4.1: Supervised machine learning (Muhammad and Yan (2015))

Supervised machine learning techniques are divided into two different types of problems. First, there are classification problems where the output is assigned to discrete variables. Second, there are regression problems where the output is assigned to continuous variables (Bishop, 2006, page 3). Since the goal of the research is to detect the arrival and departure, supervised learning including classification problems is used. The outcome of the arrival detection will be: arrived or non-arrived. The same applies to the departures: departed or non-departed. Hence, we can speak of binary classification. Some functional classification models are described in the next subsections.

4.1.2 Decision Tree

Decision Trees (DTs) can be seen as a basic model. When the data is nonlinear, DT can be used. The trees are built through split the data multiple times, starting from the root node. The data is split into subsets from this node according to certain cutoff values in the features until the terminal node (Molnar, 2019), also known as the leaf node. Every cutoff is called a decision rule. How many decision rules are taken depends on the *max_depth* parameter. When this parameter is set to high, the biggest disadvantage of DTs will occur, namely overfitting. The green line in Figure 4.2 is the result of overfitting. The DT learned from the exceptions.

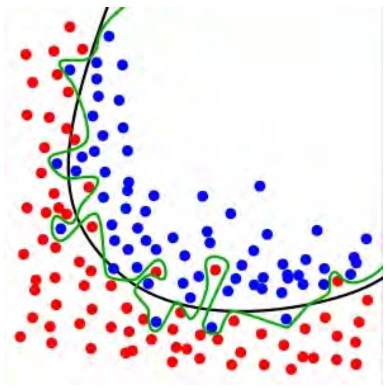


Figure 4.2: The green line represents overfitting and the black line represents a good fit

4.1.3 Random Forest

Ensemble learning is an improved machine learning technique by combining several models into one model. Bagging and boosting (Section 4.1.4) are two popular ensemble methods. A popular bagging method is the Random Forest (RF) algorithm. According to Couronné et al. (2018), RF creates data samples from the data set with bagging (bootstrap aggregation). For each sample data, a DT is trained. Every tree gives a prediction, whereby the final prediction is the class with the most votes, as in Figure 4.3. RF can handle large data sets. In addition, an advantage over the DT mentioned by Xuan et al. (2018) is that the algorithm is not that sensitive to extreme values and, therefore, not easy to overfit. However, the RF has some disadvantages as well. First, the training process requires much computational time in combination with many trees to build. Secondly, ensemble learning suffers from more difficult interpretability. The combination of all the trees is like a black box.

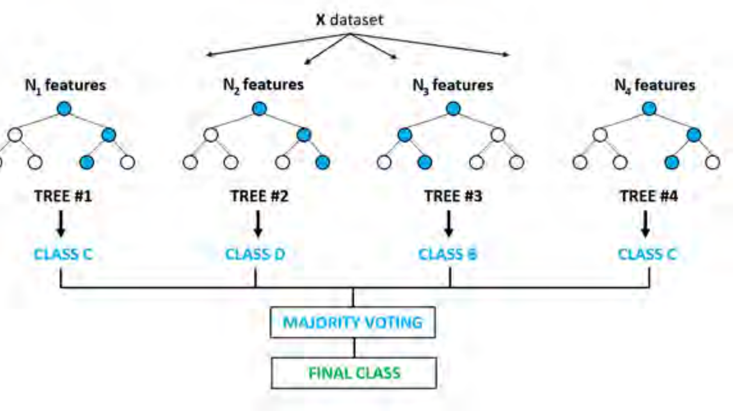


Figure 4.3: Structure of the Random Forest (Habib et al., 2020)

RF is the first model to be implemented and tested. While building the RF, there are some essential hyperparameters to tune, see Table 4.1. The column *range* denotes the tried values while tuning the hyperparameters. Note that when the hyperparameter bootstrap is set to false, the whole data set is used to build each tree. The hyperparameter *criterion* can be set with Equation (4.1), the Gini index, or the Entropy in Equation (4.2), where c is the number of classes and P_j is the probability of class j (Han et al., 2012c, page 341). The main difference between the criteria is the ranges of the interval and the computation time. The values of the Gini index are inside the interval $[0, 0.5]$, while the Entropy interval is $[0, 1]$. The computation time of the Gini index is faster since the Entropy uses the logarithm function (Aznar, 2020).

Table 4.1: The hyperparameters of the Random Forest

Hyperparameter	**Definition**	**Default**	**Range**
n_estimators	Number of trees to build	100	[20,200]
criterion	Function to measure the quality of a split	Gini	Gini,Entropy
max_depth	Maximum number of nodes in a tree	None	[10,50]
min_samples_split	Minimum number of instances required before splitting a node	2	[2,15]
min_samples_leaf	Minimum number of instances required to be at the leaf node	1	[1,5]
max_features	The number of features to consider when looking for the best split	sqrt	[2,number of features]
bootstrap	Uses bootstrap samples when building trees	True	True,False

$$Gini = 1 - \sum_{j}^{c} P_j^2. \tag{4.1}$$

$$Entropy = -\sum_{j}^{c} P_j \cdot \log_2 \cdot P_j. \tag{4.2}$$

Random Forest Algorithm

Algorithm 1 is the RF algorithm described by Hastie et al. (2009a).

```
Algorithm 1: Random Forest
for $b$=1 to $B$: do
    (a) Draw a bootstrap sample $Z^*$ of size N from the training data;
    (b) Build a random forest tree $T_b$ to the bootstrapped data, by recursively repeating the
        following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is
        reached;
          i Select $m$ features at random from the $p$ features.
         ii Pick the best features/split-point among the $m$ features.
        iii Split the node into two daughter nodes.
end

Output the ensemble of trees $\{T_b\}_1^B$;

To make a prediction at a new point $x$:
Let $\hat{C}_b(x)$ be the class prediction of the $b$th random forest tree. Then
  $\hat{C}_{rf}^B(x) = majority\ vote\{\hat{C}_b(x)\}_1^B$
```

4.1.4 Gradient Boosting

Gradient Boosting (GB) is another ensemble learning method. A difference between the bagging method RF is that GB builds one tree at a time. According to Zhang and Haghani (2015), the boosting method strategically resamples the training data to provide the most useful information for each consecutive model. This is one of the major differences between bagging and boosting. The misclassification errors of the previous models are used while building the new tree. This new tree tries to reduce the errors from the previous tree. To reduce the error, a loss function is used to determine the error. Figure 4.4 is a visualization of GB where the red bar denotes the error. Hence, the combining process of the ensemble learning of the trees is from the beginning, while the RF algorithm combines the trees at the end of the process. There are some drawbacks to the boosting technique. There is a risk of overfitting since the GB keeps trying to minimize the error. This can be avoided by tuning the *n_estimators* parameter, which denotes the number of boosting stages to perform. Another drawback is the training time, which can take a long time since all the trees are built sequentially.

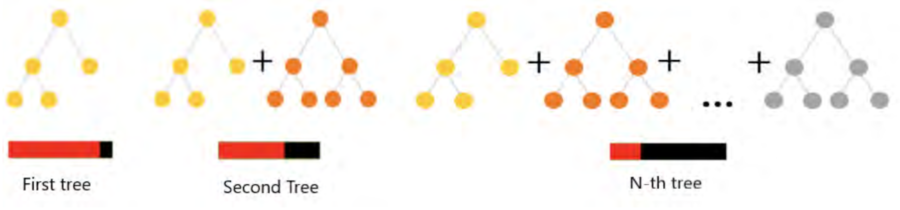


Figure 4.4: Structure of the Gradient Boosting (CatBoost, 2018)

Gradient Boosting Algorithm

The algorithm of the GB is described by Zhang and Haghani (2015). The first step is to set the objective function:

$$Objective = \min \ Loss. \tag{4.3}$$

Since we want to minimize the errors, the objective consists of the loss function, which we want to minimize. Next, the model $f_0(x)$ is initialized with a constant value:

$$f_0(x) = argmin_p \sum_{i=1}^{N} L(y_i, \rho). \tag{4.4}$$

Now we can compute the negative gradient of the loss function for all instances:

$$z_{im} = - \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}} \tag{4.5}$$

The GB continues by fitting a decision tree $g_m(x)$ to predict the targets z_{im} from covariates x_i for all training data. Whereafter, the gradient descent step size is computed:

$$p_m = argmin_p \sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + p g_m(x_i)) \tag{4.6}$$

Now we have built a new tree, the model wil be updated as follow:

$$f_m(x) = f_{m-1}(x) + p_m g_m(x), \tag{4.7}$$

where the first part shows the prediction of the previous trees and the second part represents the tree that we just built, as in Figure 4.4. This process will repeat from Equation (4.5) till (4.7) until we reached M, the number of estimators to build.

4.1.5 XGBoost

Over the years, evolution took place on tree-based algorithms. The first algorithm is the DT, followed by Bagging, RF, Boosting, GB, and XGBoost. XGBoost stands for eXtreme Gradient Boosting and is based on gradient boosting trees. As mentioned by Chen and Guestrin (2016), a regularization term is introduced by XGBoost, and the objective function from Equation (4.3) is now defined as follow:

$$Objective = min\ Loss + \Omega, \tag{4.8}$$

where Ω denotes the regularization term. This term consists of three parts as in Equation (4.9). The first term is the weight γ times the T leaves. The second term is the L1 regularization term on the weight, where α is the regularization weight, and w_j are the values of leaf j. The last term is the L2 regularization term on the weight and uses λ as the regularization weight parameter. The regularization terms are very similar. The difference is that the L1 regularization takes the absolute value of the leaves, while the L2 regularization term takes the square of the leaves.

$$\Omega = \gamma T + \alpha \sum_{j=1}^{T} |w_j| + \frac{1}{2}\lambda \sum_{j=1}^{T} W_j^2. \tag{4.9}$$

The introduced regularization term can prevent overfitting, as Mo et al. (2019) described, which is an advantage over the GB. Another advantage of XGBoost is the use of parallel and distributed computation, making the learning process faster to give a quicker modeling process. In addition to these advantages, XGBoost can handle missing values by train both routes of the split and select the best option. A disadvantage of the XGBoost is that it does not work well on small data sets, and it has many hyperparameters, which require parameter tuning to improve the performance.

XGBoost is the second model to be implemented and tested. Table 4.2 displays the hyperparameters to be tuned. Where the column *range* denotes the tried values during the tuning process. Beside these hyperparameters, the used learning objective and the evaluation metric need to be set. The objective is set on *binary* : *logistic*. This objective is the logistic regression for binary classification, whereby the output is the probability belonging to a certain class.

$$Logistic\ Regression = \frac{e^x}{1 + e^x}. \tag{4.10}$$

The output of the logistic regression is in the range of [0,1]. Thus, an output above 0.5 belongs to class 1, and otherwise, the output belongs to class 0. For the evaluation metric, the *error* is used, which is for binary classification as well.

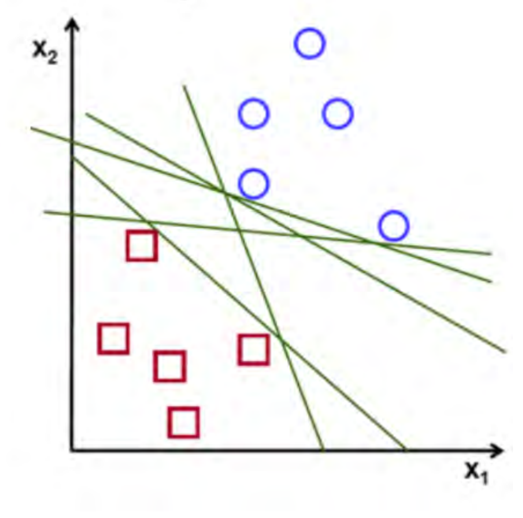
$$Error = \frac{number\ of\ wrong\ cases}{number\ of\ all\ cases}. \tag{4.11}$$

Table 4.2: The hyperparameters of the XGBoost

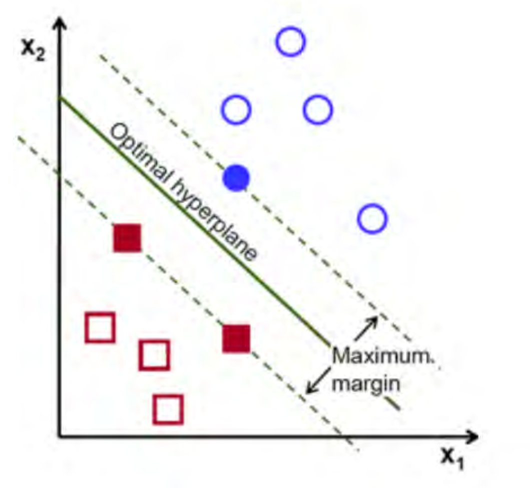
Hyperparameter	**Definition**	**Default**	**Range**
num_boost_round	Number of trees to build	-	[20,200]
eta (learning rate)	Prevents overfitting by using step size shrinkage in the update	0.3	[0.1,1]
gamma	Minimum loss reduction required to make a split	0	[1,5]
max_depth	Maximum number of nodes in a tree	6	[5,50]
min_child_weight	Minimum sum of instance weight needed in a child to make a split	1	[1,15]
max_delta_step	Maximum delta step we allow each leaf output to be	0	[0,10]
subsample	Takes a subset of the data	1	[0.5,1]
colsample_bytree	Ratio of features when constructing each tree	1	[0.1,1]
colsample_bylevel	Ratio of features for each level in the tree	1	[0.1,1]
colsample_bynode	Ratio of feature for each node in the tree	1	[0.1,1]
alpha	L1 regularization	0	[0,1]
lambda	L2 regularization	1	[0,1]

4.1.6 Support Vector Machines

Support Vector Machines (SVM) can be used for both linear and nonlinear data sets. First, we focus on the linear SVM. The goal of the SVM described by Hastie et al. (2009b), is to find the optimal hyperplane between two classes as in Figure 4.5a. The hyperplane dimensionality equals the number of features minus one. The maximum margin in Figure 4.5b is used to find the optimal hyperplane. The maximum margin is the biggest distance between the data points of the two distinctive classes.



(a) Find optimal hyperplane



(b) Maximum margin

Figure 4.5: Hyperplane SVM (Ippolito, 2019)

When dealing with nonlinear data, the kernel trick can be applied (Witten et al., 2017). The kernel trick transformes nonlinear data into linear data. The most common kernel functions are the polynomial in Equation (4.12), radial basis function (RBF) in Equation (4.13), and the sigmoid in Equation (4.14) (Han et al., 2012b). How these kernel functions are applied in the SVM is explained during the description of the SVM algorithm below.

$$Polynomial : K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^h \tag{4.12}$$

$$RBF : K(\mathbf{x}_i, \mathbf{x}_j) = e^{-||\mathbf{x}_i - \mathbf{x}_j||^2 / 2\sigma^2} \tag{4.13}$$

$$Sigmoid : K(\mathbf{x}_i, \mathbf{x}_j) = tanh(\kappa \mathbf{x}_i \cdot \mathbf{x}_j - \delta) \tag{4.14}$$

SVM has the advantage it can still be used on data sets with high dimensionality, even when the dimensional is higher than the number of instances (Auria and Moro, 2008). A major drawback of using the SVM model is the required training time for large data sets. In addition, when the number of features is much greater than the number of samples, choosing the appropriate kernel function and regularization term is crucial to avoid overfitting. Moreover, the SVM uses an expensive 5-fold cross validation to provide probability estimates (Pedregosa et al., 2011). Therefore, the preference is to use SVM models on smaller data sets.

The third model to be implemented and tested is the SVM. The hyperparameters in Table 4.3 will be tuned to receive the best results using the SVM. A misclassification receives a higher penalty with a higher penalty term C and will be tuned in the range between 1 and 100.

Table 4.3: The hyperparameters of the Support Vector Machine

Hyperparameter	**Definition**	**Default**	**Range**
C	Penalty term	1.0	[1,100]
kernel	Kernel type	RBF	Poly,RBF,Sigmoid
gamma	Kernel coefficient	Scale	Scale,Auto

Support Vector Machine Algorithm

The SVM algorithm is described by Kecman and Wang (2005), Hastie et al. (2009b) and Cristianini and Ricci (2008) where the notation of Kecman and Wang is used. We train a data set of N pairs and p features: $D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N) | x_i \in R^p, y_i \in \{-1, 1\}\}$. The hyperplane is defined as:

$$d(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}^T \mathbf{x} + b = \sum_{i=1}^{n} w_i x_i + b, \tag{4.15}$$

where b is the scalar and is called the bias. After training the model, the weights are obtained, and unseen data, $\mathbf{x}_p$, can be feed to the model. The SVM produces output 0 according to an *indicator function* given by:

$$iF = o = sign(d(\mathbf{x}_p, \mathbf{w}, b)), \tag{4.16}$$

where the standard notation for the output from the SVM equals 0. Here from follows if $d(\mathbf{x}_p, \mathbf{w}, b) > 0$, the pattern $\mathbf{x}_p$ belongs to class 0, else the pattern belongs to class 1. To find the optimal hyperplane, the margin $M = \frac{2}{||\mathbf{w}||}$ should be maximized. Minimizing the weights $\mathbf{w}^T \mathbf{w} = \sum_{i=1}^{n} w_i^2$ leads to the maximization of the margin M. Hence, the learning problem is

$$\min \ \frac{1}{2} \mathbf{w}^T \mathbf{w} \ \textit{subject to} \ y_i [\mathbf{w}^T \mathbf{x}_i + b] \geq 1, \ i = 1, \cdots, N. \tag{4.17}$$

The classes in the feature space may overlap each other. To be able optimizing M with overlapping classes, slack variables, $\xi = (\xi_1, \xi_2, \cdots, \xi_N)$, should be added. With these slack variables, it is allowed to be on the wrong side of the margin for some points, also known as the soft margin. We can rewrite Equation (4.17) as follow:

$$\min \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i^k \; subject\ to \begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i, \; for\ y_i = 1, \xi_i \geq 0 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i, \; for\ y_i = -1, \xi_i \geq 0 \end{cases} \tag{4.18}$$

This is the defined support vector classifier for the non-separable case, where C is the cost parameter (the penalty term in Table 4.3). In the case of the separable case, C is set to infinity. Cross validation is used to determine how many misclassifications and observations are allowed inside of the soft margin to get the best classification. Equation (4.18) is a convex programming problem, which can be solved with the Lagrange primal function

$$Lp(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\{y_i[\mathbf{w}^T\mathbf{x}_i + b] - 1 + \xi_i\} - \sum_{i=1}^{N}\beta_i\xi_i, \tag{4.19}$$

where α_i and β_i are the Lagrange multipliers. The Lagrangian function has to be minimized with respect to $\mathbf{w}$, b and ξ, and maximized with respect to the Lagrange multipliers. This problem can be solved in a dual space. To minimize $\mathbf{w}$, b and ξ_i, we take the following derivatives and set these derivatives to zero. We obtain the following results:

$$\frac{\partial Lp}{\partial \mathbf{w}_0} = 0, \; i.e., \; \mathbf{w}_0 = \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i \tag{4.20}$$

$$\frac{\partial Lp}{\partial \mathbf{b}_0} = 0, \; i.e., \; \sum_{i=1}^{N}\alpha_i y_i = 0 \tag{4.21}$$

$$\frac{\partial Lp}{\partial \xi_{i0}} = 0, \; i.e., \; \alpha_i + \beta_i = C \tag{4.22}$$

The next step for solving the problem in the dual space is to satisfies the Karush-Kuhn-Tucker (KKT) conditions below,

$$\alpha_i\{y_i[\mathbf{w}^T\mathbf{x}_i + b] - 1 + \xi_i\} = 0, \; i = 1\cdots, N \tag{4.23}$$

$$\beta_i\xi_i = (C - \alpha_i)\xi_i = 0, \; i = 1\cdots, N \tag{4.24}$$

Substituting (4.20), (4.21) and (4.22) into (4.19) will lead to the Lagrangian Wolfe dual objective function:

$$\begin{gathered} L_d(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j \\ subject\ to\ 0 \leq \alpha_i \leq C\ and\ \sum_{i=1}^{N}\alpha_i y_i = 0 \end{gathered} \tag{4.25}$$

The Lagrangian Wolfe objective function only depends on the Lagrangian multipliers, which is a advantage since it is easier to be solved. Since we are dealing with nonlinear data, the kernel trick can be used from this point. Instead of the Lagrange Wolfe dual above, the following objective function will be used with the same constraint as in Equation (4.25):

$$L_d(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j), \tag{4.26}$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ is one of the kernel functions from Equation (4.12), (4.13), or (4.14) As described by (Han et al., 2012b, page 413) the nonlinear data is transformed into a higher dimensional space. Hereby, nonlinear mapping is used. After transforming the data into a higher dimensional space, a linear separating hyperplane can be found.

4.1.7 Artificial Neural Network

The first created Neural Network (NN) is invented by Rosenblatt and is called the Perceptron, which can be used for binary classification problems (Rosenblatt, 1958). The Perceptron contains an artificial neuron, which will be explained later. The input layer consists of N neurons (the input features), and the output layer is for binary classification (1 if it is truth, else 0) and consists of one neuron. The Perceptron has evolved, nowadays multiple artificial neurons with multiple hidden layers can be used and are called a Multi-Layer Perceptron (MLP). The MLP consist of minimal three layers: one input layer, minimum one hidden layer, and one output layer. A visualization of a two-layer (the input layer is not counted) MLP with four neurons in the hidden layer is visualized in Figure 4.6.

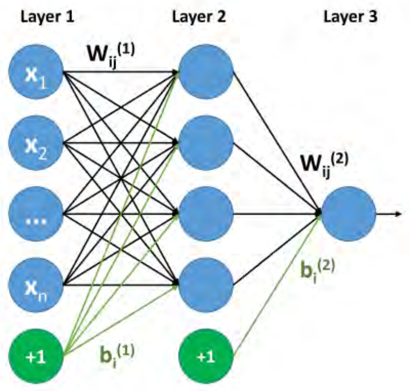


Figure 4.6: Example of a MLP with n input features, one hidden layer with four neurons and the output layer with one neuron

The input layer of the MLP consist of the input features, also known as the input signals. The input layer passes the features onto the next layer. In the next layer, the first hidden layer, the artificial neurons will apply different transformations to the data. The operation of these artificial neurons is visualized in Figure 4.7. The input values will be multiplied, each with its own weight

(W), modified during the learning process. All these values will be added together with the bias b. The final operation of the neuron is applying the activation function. Equation (4.27) gives the output (y) of a neuron, where ϕ is the activation function (Baldeschi et al., 2019). The different activation functions and their behavior will be explained later. If the MLP consists of multiple hidden layers, the input of each artificial neuron in the current layer is the output of all artificial neurons of the previous layer. Hence, we speak of a *fully connected* network since all the neurons in a layer are connected to all neurons in the next layer. The same operations of the artificial neurons will be applied in all neurons of the remaining layers. The MLP is a *feedforward* neural network since the information is only passed forward to the next layer. This first phase of the training process from the input layer to the output layer is called *forward propagation.*

$$y = \phi(\sum_{j=1}^{n} w_i x_i + b). \tag{4.27}$$

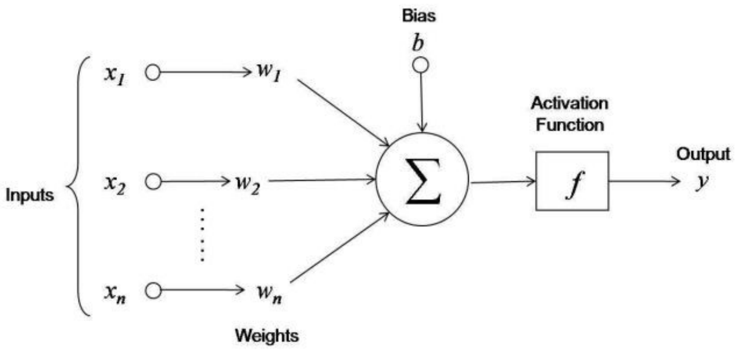


Figure 4.7: Operations done by a neuron (Arnx, 2019)

Backpropagation

The output of the MLP is affected by the weights (W) and the biases (b) and need therefore be optimized. To find the optimal weights and biases, we first need to define a loss function. The lower the loss, the better the model. Thus, we can speak of a minimization problem. Since we are dealing with a binary classification problem, the error of the predictions is determined with the binary cross-entropy loss function (Ho and Wookey, 2020):

$$Binary\ Cross - Entropy = \frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i)), \tag{4.28}$$

where N is the number of training instances, y_i represents the actual class, and $\log p(y_i)$ and $1 - \log p(y_i)$ is the probability of class 1 and 0, respectively. The task of the loss function is to compare the probabilities with the truth labels and calculate the penalty for any deviation between this truth label and the output of the MLP (Ho and Wookey, 2020). Now we know the error of our prediction, we will update the weights and biases by propagating the error back, which is called *backpropagation.*

The gradient descent technique is used to get the loss as close as possible to zero next time. As described by Torres (2020), the gradient descent technique changes the weights in small increments. Therefore, the derivative of the loss function with respect to the weights is determined, which tells us the direction we need to move towards to reach the minima, as in Figure 4.8. The step size of the gradient direction is determined by the learning rate and needs to be optimized during the hyperparameter optimization. Zhang (2019) described that the learning rate should not be set too large since we may miss the minimum, but it should not be set too small either so we won't get stuck in a local minimum. The update rule is in Equation (4.29), where α is the learning rate. After a specific batch size, the parameters are updated due to the gradient descent. There are different batch sizes of the gradient descent mentioned by Zhang:

- **Batch gradient descent**: all data is trained at once. All the instances are used to calculate the gradient. A drawback is a high risk of getting stuck at a local minimum.
- **Stochastic gradient descent**: one instance is trained at once and is used to calculate the gradient. A drawback is a high variance in the model parameters and the computation time since a lot more iterations are needed.
- **Mini-batch gradient descent**: instead of all data or one instance, N random instances are trained each iteration. The best size of N should be set while optimizing the hyperparameters, called *batch_size*.
- **Momentum**: the momentum is an extension of the stochastic gradient descent. It helps to converge faster.

The number of *epochs* denotes the number of times the processes forward and back propagation is carried out on all the data.

$$W_{ij} = W_{ij} - \alpha \frac{\partial Error}{\partial W_{ij}} \tag{4.29}$$

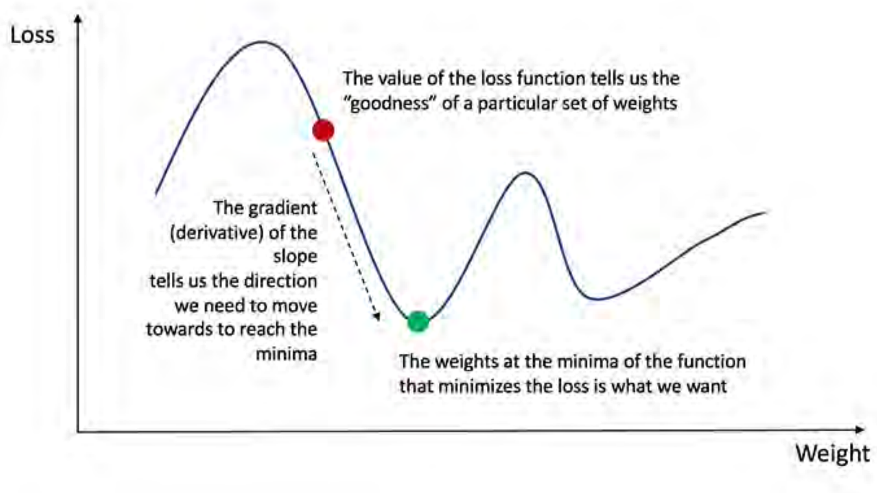
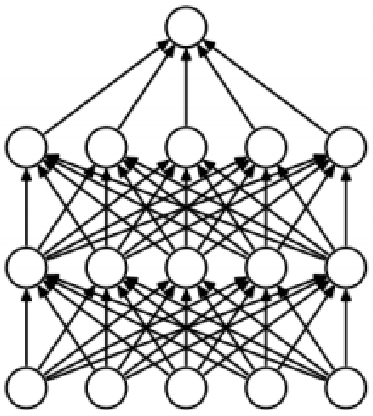
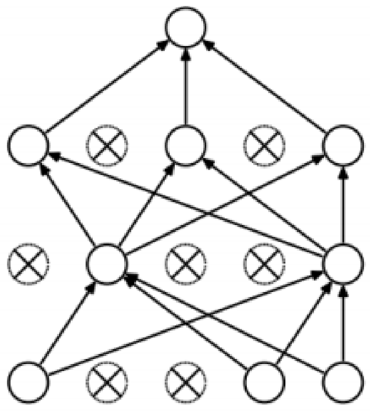

Loss
The value of the loss function tells us the "goodness" of a particular set of weights
The gradient (derivative) of the slope tells us the direction we need to move towards to reach the minima
The weights at the minima of the function that minimizes the loss is what we want
Weight

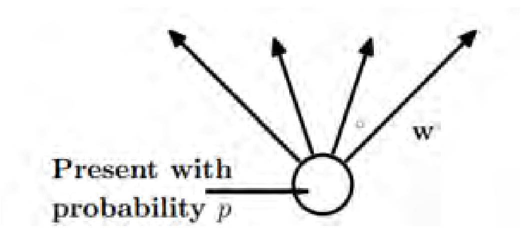
Figure 4.8: Gradient descent algorithm (Loy, 2018)

Dropout

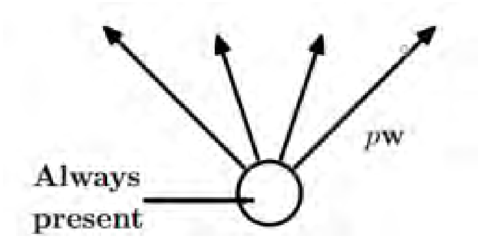
A common problem in MLP is overfitting due to the many hyperparameters that can be tuned. To avoid this problem, the technique *dropout* can be used (Srivastava et al., 2014). Using dropouts does not use all neurons during the training phase, as shown in Figures 4.9a and 4.9b. The dropout of a neuron is determined by assigning a random probability p to the network. With probability p, the neuron will remain in the network, and with probability $1 - p$, the neuron will be removed. The dropout rate can be applied on the input layer and the hidden layers. During the test phase, all neurons remain in the network, but the weights are multiplied by p so that the actual output during the test phase is the same as the expected output from the training phase. This is visualized in Figures 4.9c and 4.9d. In addition that dropout can prevent overfitting, Srivastava et al. mentioned dropout is a technique that can try many different architectures of the MLP more efficiently.


(a) Standard MLP


(b) MLP after applying dropout



(c) Neuron at training time with probability p



(d) Neuron at test time with probability p

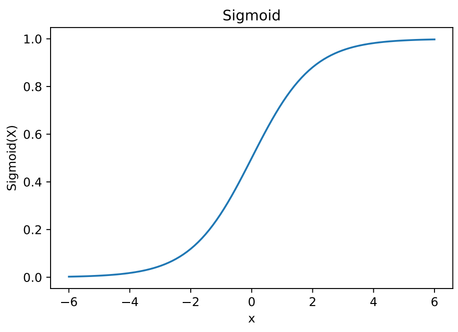
Figure 4.9: Dropout Multi-Layer Perceptron (Srivastava et al., 2014)

Hyperparameters

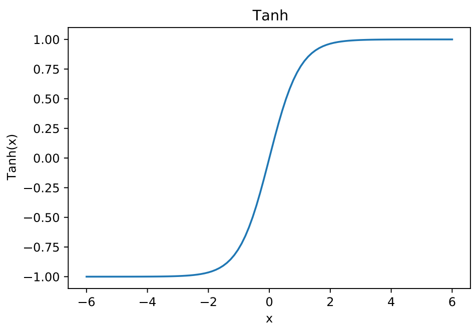
Activation functions are used to control the outputs of the MLP as described by Nwankpa et al. (2020). To get the desired output, the correct activation function should be used. With an activation function, it is possible to receive nonlinearity output. The different activation functions are in Table 4.4, with a visualization in Figure 4.10.

Table 4.4: Activation functions

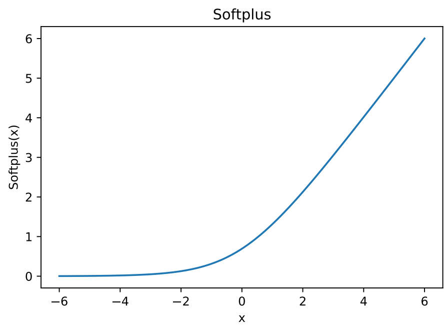
Activation function	**Function**	**Range**
sigmoid	$\phi(x) = \frac{1}{1+e^{-x}}$	(0,1)
tanh	$\phi(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	(-1,1)
softplus	$\phi(x_i) = log(1+e^{x})$	[0,inf)
ReLU	$\phi(x) = \max(0, x)$	[0,inf)



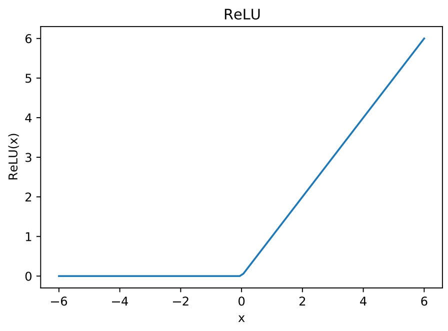
(a) Sigmoid



(b) Tanh



(c) Softplus



(d) ReLU

Figure 4.10: Activation functions

The most successful and widely used activation function in the hidden layers is the Rectified Linear Unit (ReLU) because of the better performance and faster learning compared to other activation functions (Nwankpa et al., 2020; Ramachandran et al., 2017). Therefore, the ReLU will be used as an activation function in the hidden layers in this research. Since we are dealing with a binary classification problem, the sigmoid activation function is used on the output layer. The sigmoid output is between 0 and 1, which will be interpreted as the probability that the input has the positive class. For optimizing the weights, the Adam optimizer will be used. This is an optimizer that works well for many different data, according to Kingma and Ba (2014). The remaining hyperparameters to tune are in Table 4.5. Kandel and Castelli (2020) recommend using a power of 2 for the batch size parameter. Therefore the values 4, 8, 16, 32, and 64 are tried for optimizing the *batch_size* parameter.

Table 4.5: The hyperparameters of the Multi-Layer Perceptron

Hyperparameter	Definition	Tried Values
epochs	The number of cycles through the full training data set	10 till 100 with steps of 5
batch_size	The number of samples through the NN before updating the weights	4,8,16,32,64
hidden_layers	Number of layers between the input and the output layer	1,2
nodes_hidden_layer1	Number of nodes in hidden layer 1	8,16,32,64
nodes_hidden_layer2	Number of nodes in hidden layer 2	0,8,16,32,64
learning_rate	Step size of updating the weights	0.01, 0.001, 0.0001
dropout_input_layer	The dropout rate in the input layer	0,0.10,0.20
dropout_hidden_layer_1	The dropout rate in hidden layer 1	0,0.10,0.20
dropout_hidden_layer_2	The dropout rate in hidden layer 2	0,0.10,0.20

Advantage and Disadvantage

The MLP has some advantages and disadvantages described by Tu (1996). An advantage of the MLP is that it can be trained by both numeric and categorical input and output features. Although, some transformation of the data is needed, which will be explained in Section 5.5.1. Another advantage is that the MLP can detect complex nonlinear relationships between independent and dependent features.

One of the major drawbacks is the unexplained behavior of the network. Its behavior is like a black box. Furthermore, developing a neural network can be time-consuming since of the many hyperparameters to tune. In addition, MLP is prone to overfitting. Moreover, training an MLP is computationally expensive.

4.2 Performance Metrics

Performance metrics are used to evaluate and compare the performance of the models. Although the accuracy and the error rate are widely used and are easy to calculate and interpret, it has some drawbacks in imbalanced data sets (Fernández et al., 2018). The majority class can be predicted well in imbalanced data, while the minority class is hard to predict correctly. In this case, a high accuracy (or a low error rate) is still obtained, which is not a good indicator. Therefore, the following performance metrics will be used.

Confusion Matrix

As Han et al. (2012c) described, the confusion matrix is a useful tool for analyzing how well the model predicts the correct labels. A confusion matrix consists of four blocks, as shown in Figure 4.11. The meaning of the blocks in the context of this research are as follow:

- **True Negatives** (TN): A non-arrival is predicted as a non-arrival
- **False Positive** (FP): A non-arrival is predicted as an arrival
- **False Negative** (FN): An arrival is predicted as a non-arrival
- **True Positive** (TP): An arrival is predicted as an arrival

The meaning of the blocks is based on the arrival detection model. However, for the departures model, non-arrival and arrival are replaced by non-departure and departure, respectively.

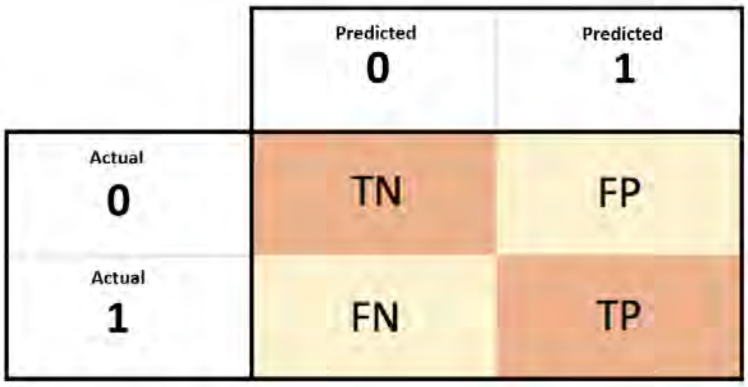


Figure 4.11: Illustration of a confusion matrix (Bhagwat et al., 2019)

Precision

Precision measures the fraction of actually correct positive identifications. In the context of this research, it is the fraction of the predicted arrivals (departures) that are actually arrivals (departures). The formula of the precision measure is:

$$Precision = \frac{TP}{TP + FP}. \tag{4.30}$$

Recall

Recall measures the fraction of actual positives that are identified correctly. In this research, it is the fraction of the total arrivals (departures) predicted as arrival (departure). The recall formula is:

$$Recall = \frac{TP}{TP + FN}. \tag{4.31}$$

F1-score

For both precision and recall, the perfect score equals one. However, the precision and recall are related, and therefore it will not be possible to get a perfect score for both measures. When the precision increases, the recall will decrease, and vice versa. An alternative way to use precision and recall is to combine them into a single measure. This is the approach of the F1 measure (Han et al., 2012c).

$$F1 = \frac{2 \times precision \times recall}{precision + recall}. \tag{4.32}$$

4.3 Cross Validation

Cross Validation (CV), also known as *k*-fold CV, is one of the most commonly used data resampling methods to tune the hyperparameters, according to Berrar (2018). When CV is used, the training data is split into *k* approximately equal partitions, each, in turn, is used for testing, and the remainder *k-1* part is used for training (Witten et al., 2011). The testing part during CV is called the validation set. The validation set differs from the test set. During CV, the test data is not used at all. This data will remain unseen until the final models will be tested. An example of a 5-fold CV is visualized in Figure 4.12. The four green folds per split are the training data, where the blue fold is the validation data. The same combination of hyperparameters is trained five times on the different compositions of the folds.

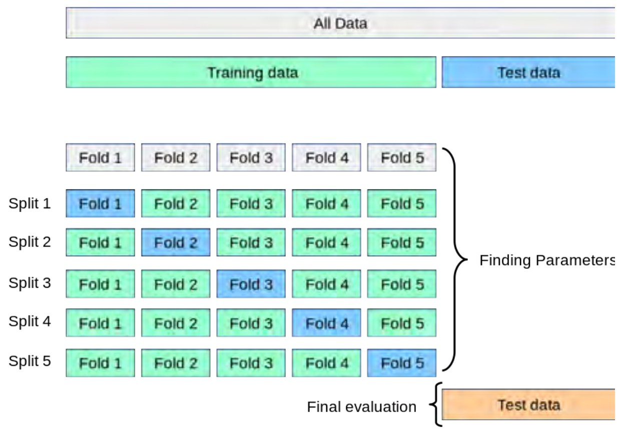


Figure 4.12: Illustration of a 5-fold cross validation (Pedregosa et al., 2011)

4.4 Hyperparameter Tuning

The discussed models have multiple hyperparameters. These hyperparameters help with the learning process and need to be tuned. Since there are many different combinations of hyperparameters, tuning manually is time-consuming. In addition, there is a chance that the best combination of hyperparameters is missed. Therefore, tuning will be performed by random search and grid search.

4.4.1 Random Search and Grid Search

Random search sets up a grid with values for the hyperparameters. Then, for each of the n-iterations, random combinations of hyperparameters are selected to train the model and evaluate. Random search is more efficient for hyperparameter tuning than grid search when there is a high dimension of hyperparameters mentioned by Bergstra and Bengio (2012).

Another manner of selecting the best combination of hyperparameters is grid search. The main difference between random and grid search is that the first method will not train and evaluate all the hyperparameter combinations while the second method does. Therefore, searching for the best hyperparameters using grid search will require an intensive and exhausting computation time. Grid search is reliable in a low dimensional of hyperparameters (Bergstra and Bengio, 2012).

Yu and Zhu (2020) recommended using random search first to rapidly reduce the search space, followed by a grid search to get a finer result. Therefore, this research uses both random and grid search. First, 100 combinations of hyperparameters are trained and evaluated using random search with a 5-fold CV. Next, a small grid is set up with hyperparameter values around the best hyperparameters after random search. Finally, grid search with a 5-fold CV is applied to obtain the best combination of hyperparameters for the final models.

4.5 Summary

Several supervised machine learning models are explained in this chapter, and four of them will be implemented and tested later in this research. First of all, the RF is selected as machine learning model. The RF builds multiple trees, where the final prediction is the class with the most votes. The RF is not that sensitive to extreme values and thus not easy to overfit.

Secondly, the XGBoost is used. The XGBoost creates multiple trees as well. The major difference is that the XGBoost uses the misclassification of the previous tree while building the new tree. While building that new tree, it will attempt to reduce the error of the prior tree. A regularization term is used while building the XGBoost.

Thirdly, we have the SVM. The SVM finds a hyperplane between two classes which maximizes the margin between the data points of the two distinctive classes. An advantage of the SVM is that it can still be used on data sets with high dimensionality, even when the dimensional is higher than the number of instances. A drawback is the required training time for large data sets.

Fourthly, we have the MLP, which consists of one or more hidden layers. A fully connected network is created since the output of each artificial neuron of the previous layer is the input for all artificial neurons of the current layer. During the backpropagation, the weights will be optimized, which will reduce the error. The MLP consists of many hyperparameters which need to be tuned.

The performance of the models is evaluated and compared with the use of performance metrics. The F1 tries to optimize both precision and recall.

To receive the best performances of the models and prevent overfitting, the hyperparameters will be tuned. Tuning these hyperparameters will be performed by random search followed by a grid search including 5-fold cross validation.

Chapter 5

Data

Confidential

Chapter 6

Results

Confidential

Chapter 7

Conclusion

Confidential

Appendix A

Transitions FSM Model

Confidential

Appendix B

Parameters Arrivals

The hyperparameters for the different models for the arrivals are determined with a 5-fold CV random search, followed by a 5-fold CV grid search. The *random_state* is set on 7 for all model types. With this setting, the results are reproducible.

Random Forest

The results of the RF hyperparameters for all the four features sets are in Table B.1.

Table B.1: The selected hyperparameters of the RF for all arrival feature sets

Hyperparameter	**Feature set 1**	**Feature set 2**	**Feature set 3**	**Feature set 4**
n_estimators	100	80	150	35
criterion	entropy	entropy	entropy	entropy
max_depth	43	25	20	22
min_samples_split	3	12	6	12
min_samples_leaf	2	1	2	1
max_features	8	19	17	17
bootstrap	True	True	True	True

XGBoost

The results of the XGBoost hyperparameters for all the four features sets are in Table B.2.

Table B.2: The selected hyperparameters of the XGBoost for all arrival feature sets

Hyperparameter	Feature set 1	Feature set 2	Feature set 3	Feature set 4
num_boost_round	35	66	90	45
max_depth	15	15	20	15
gamma	3	3	3	3
min_child_weight	7	3	3	7
eta	0.30	0.25	0.20	0.30
max_delta_step	1	1	1	1
subsample	1	1	1	1
colsample_bytree	1	1	0.5	1
colsample_bylevel	1	1	0.8	1
colsample_bynode	1	1	0.5	1
alpha	0	0	0	0
lambda	1	1	0.5	1

Support Vector Machine

The results of the SVM hyperparameters for all the four features sets are in Table B.3.

Table B.3: The selected hyperparameters of the SVM for all arrival feature sets

Hyperparameter	Feature set 1	Feature set 2	Feature set 3	Feature set 4
kernel	RBF	RBF	RBF	RBF
gamma	scale	auto	auto	auto
C	25	15	5	20

Multi-Layer Perceptron

The results of the MLP hyperparameters for all the four features sets are in Table B.4.

Table B.4: The selected hyperparameters of the MLP for all arrival feature sets

Hyperparameter	Feature set 1	Feature set 2	Feature set 3	Feature set 4
Epochs	50	80	90	100
Batch size	64	64	64	64
# hidden layers	2	2	2	1
# nodes hidden layer 1	16	16	64	32
# nodes hidden layer 2	8	8	16	0
Learning rate	0.001	0.001	0.001	0.001
Dropout input layer	0.0	0.0	0.0	0.0
Dropout hidden layer 1	0.0	0.0	0.2	0.0
Dropout hidden layer 2	0.0	0.0	0.0	0.0

Appendix C

Detected Arrivals

Confidential

Appendix D

Parameters Departures

The hyperparameters for the different models for the departures are determined with a 5-fold CV random search, followed by a 5-fold CV grid search. The *random_state* is set on 7 for all model types. With this setting, the results are reproducible.

Random Forest

The results of the RF hyperparameters for all the four features sets are in Table D.1.

Table D.1: The selected hyperparameters of the RF for all departure feature sets

Hyperparameter	**Feature set 1**	**Feature set 2**	**Feature set 3**	**Feature set 4**
n_estimators	90	90	50	130
criterion	entropy	entropy	gini	entropy
max_depth	25	25	40	20
min_samples_split	11	2	5	2
min_samples_leaf	1	1	1	1
max_features	3	15	6	19
bootstrap	True	True	True	True

XGBoost

The results of the XGBoost hyperparameters for all the four features sets are in Table D.2.

Table D.2: The selected hyperparameters of the XGBoost for all departure feature sets

Hyperparameter	**Feature set 1**	**Feature set 2**	**Feature set 3**	**Feature set 4**
num_boost_round	45	60	70	55
max_depth	5	19	25	18
gamma	3	2	1	1
min_child_weight	7	2	1	2
eta	0.30	0.20	0.25	0.20
max_delta_step	1	1	1	1
subsample	1	1	1	1
colsample_bytree	1	1	0.5	1
colsample_bylevel	1	1	1	1
colsample_bynode	1	1	1	1
alpha	0	0	0	0
lambda	1	1	0.6	1

Support Vector Machine

The results of the SVM hyperparameters for all the four features sets are in Table D.3.

Table D.3: The selected hyperparameters of the SVM for all departure feature sets

Hyperparameter	**Feature set 1**	**Feature set 2**	**Feature set 3**	**Feature set 4**
kernel	RBF	RBF	RBF	RBF
gamma	auto	scale	auto	auto
C	5	13	45	15

Multi-Layer Perceptron

The results of the MLP hyperparameters for all the four features sets are in Table D.4.

Table D.4: The selected hyperparameters of the SVM for all departure feature sets

Hyperparameter	**Feature set 1**	**Feature set 2**	**Feature set 3**	**Feature set 4**
Epochs	50	50	55	40
Batch size	64	32	64	32
# hidden layers	2	1	2	2
# nodes hidden layer 1	64	32	64	32
# nodes hidden layer 2	16	0	16	16
Learning rate	0.001	0.0001	0.001	0.001
Dropout input layer	0.0	0.0	0.0	0.0
Dropout hidden layer 1	0.1	0.0	0.0	0.0
Dropout hidden layer 2	0.0	0.0	0.0	0.0

Appendix E

Detected Departures

Confidential

References

Catboost enables fast gradient boosting on decision trees using gpus, 2018. URL `https://catboost.ai/news/catboost-enables-fast-gradient-boosting-on-decision-trees-using-gpus`. Last visited on June 8, 2021.

J. Alzubi, A. Nayyar, and A. Kumar. Machine learning from theory to algorithms: An overview. *Journal of Physics: Conference Series*, 1142:012012, nov 2018. doi: 10.1088/1742-6596/1142/1/012012. URL `https://doi.org/10.1088/1742-6596/1142/1/012012`.

A. Arnx. First neural network for beginners explained (with code), 2019. URL `https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-\protect\@normalcr\relax4cfd37e06eaf`. Last visited on June 21, 2021.

L. Auria and R. Moro. Support vector machines (svm) as a technique for solvency analysis. *SSRN Electronic Journal*, 1, 02 2008. doi: 10.2139/ssrn.1424949.

P. Aznar. Decision trees: Gini vs entropy, 2020. URL `https://quantdare.com/decision-trees-gini-vs-entropy/`. Last visited on May 20, 2021.

A. Baldeschi, R. Margutti, and A. Miller. Deep learning: a new definition of artificial neuron with double weight. *CoRR*, abs/1905.04545, 2019. URL `http://arxiv.org/abs/1905.04545`.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(null):281–305, Feb. 2012. ISSN 1532-4435.

D. Berrar. Cross-validation. *Encyclopedia of Bioinformatics and Computational Biology*, 01 2018. doi: 10.1016/B978-0-12-809633-8.20349-X.

R. Bhagwat, M. Abdolahnejad, and M. Moocarme. *Applied Deep Learning with Keras*. Packt Publishing, 2019.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, ACM*, pages 785–794, 2016.

R. Couronné, P. Probst, and A. Boulesteix. Random forest versus logistic regression: a large-scale benchmark experiment. *BMC Bioinormatics*, 19:270, 2018. doi: https://doi.org/10.1186/s12859-018-2264-5.

N. Cristianini and E. Ricci. Support vector machines. In *Encyclopedia of Algorithms*, pages 928–932. Springer US, Boston, MA, 2008. doi: 10.1007/978-0-387-30162-4_415. URL https://doi.org/10.1007/978-0-387-30162-4_415.

A. Dey. Machine learning algorithms: a review. *International Journal of Computer Science and Information Technologies*, 7(3):1174–1179, 2016.

A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera. *Performance Measures*, pages 47–61. Springer International Publishing, Cham, 2018. doi: 10.1007/978-3-319-98074-4_3. URL https://doi.org/10.1007/978-3-319-98074-4_3.

M. T. Habib, A. Majumder, R. N. Nandi, F. Ahmed, and M. S. Uddin. A comparative study of classifiers in the context of papaya disease recognition. In *Proceedings of International Joint Conference on Computational Intelligence*, pages 417–429. Springer Singapore, 2020.

J. Han, M. Kamber, and J. Pei. 1 - introduction. In J. Han, M. Kamber, and J. Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 1–38. Morgan Kaufmann, Boston, third edition edition, 2012a. doi: https://doi.org/10.1016/B978-0-12-381479-1.00001-0.

J. Han, M. Kamber, and J. Pei. 9 - classification: Advanced methods. In J. Han, M. Kamber, and J. Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 393–442. Morgan Kaufmann, Boston, third edition edition, 2012b. doi: https://doi.org/10.1016/B978-0-12-381479-1.00009-5.

J. Han, M. Kamber, and J. Pei. 8 - classification: Basic concepts. In J. Han, M. Kamber, and J. Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 327–391. Morgan Kaufmann, Boston, third edition edition, 2012c. doi: https://doi.org/10.1016/B978-0-12-381479-1.00008-3.

T. Hastie, R. Tibshirani, and J. Friedman. *Random Forests*, pages 587–604. Springer New York, New York, NY, 2009a. doi: 10.1007/978-0-387-84858-7_15. URL https://doi.org/10.1007/978-0-387-84858-7_15.

T. Hastie, R. Tibshirani, and J. Friedman. *Support Vector Machines and Flexible Discriminants*, pages 417–458. Springer New York, New York, NY, 2009b. doi: 10.1007/978-0-387-84858-7_12. URL https://doi.org/10.1007/978-0-387-84858-7_12.

Y. Ho and S. Wookey. The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling. *IEEE Access*, 8:4806–4813, 2020. doi: 10.1109/ACCESS.2019.2962617.

P. P. Ippolito. Svm: Feature selection and kernels, 2019. URL https://towardsdatascience.com/svm-feature-selection-and-kernels-840781cc1a6c. Last visited on April 8, 2021.

I. Kandel and M. Castelli. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, 6(4):312–315, 2020. doi: https://doi.org/10.1016/j.icte.2020.04.010.

V. Kecman and L. Wang. *Support Vector Machines – An Introduction*, pages 1–47. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. doi: 10.1007/10984697_1. URL https://doi.org/10.1007/10984697_1.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

D. Liu, J. Sun, and S. Wang. Bustime: Which is the right prediction model for my bus arrival time? *Ieee Internation Conference on Big Data Analytics*, pages 180–185, 2020. doi: 10.1109/ICBDA49040.2020.9101265.

J. Loy. How to build your own neural network from scratch in python, 2018. URL `https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-\python-68998a08e4f6`. Last visited on June 23, 2021.

H. Mo, H. Sun, J. Liu, and S. Wie. Developing window behavior models for residential buildings using xgboost algorithm. *Energy & Buildings*, 205, 2019. doi: https://doi.org/10.1016/j.enbuild.2019.109564.

C. Molnar. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable.* Lulu, 1st edition, March 24, 2019.

I. Muhammad and Z. Yan. Supervised machine learning approaches: A survey. *ICTACT Journal on Soft Computing*, 5(3), 2015. doi: 10.21917/ijsc.2015.0133.

C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 12 2020.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. URL `http://arxiv.org/abs/1710.05941`.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

D. Sun, H. Luo, L. Fu, W. Liu, X. Liao, and M. Zhao. Predicting bus arrival time on the basis of global positioning system data. *Transportation Research Record*, 2034(1):62–72, 2007. doi: 10.3141/2034-08. URL `https://doi.org/10.3141/2034-08`.

L. Ting, Z. Xinqei, X. Guangluan, F. Kun, and R. Wenjuan. An improved dbscan algorithm to detect stops in individual trajectories. *Isprs International Journal of Geo-Information*, 6(3), 2017. doi: https://doi.org/10.3390/ijgi6030063.

J. Torres. Learning process of a deep neural network, 2020. URL `https://towardsdatascience.com/learning-process-of-a-deep-neural-network-5a9768d7a651`. Last visited on July 4, 2021.

J. V. Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11):1225–1231, 1996. doi: https://doi.org/10.1016/S0895-4356(96)00002-9.

I. H. Witten, E. Frank, and M. A. Hall. Chapter 5 - credibility: Evaluating what's been learned. In *Data Mining: Practical Machine Learning Tools and Techniques (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 147–187. Morgan Kaufmann, Boston, third edition edition, 2011. doi: https://doi.org/10.1016/B978-0-12-374856-0.00005-5.

I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques.* The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, Boston, fourth edition edition, 2017.

S. Xuan, G. Liu, and Z. Li. Refined weighted random forest and its application to credit card fraud detection. In X. Chen, A. Sen, W. W. Li, and M. T. Thai, editors, *Computational Data and Social Networks*, pages 343–355, Cham, 2018. Springer International Publishing.

X. Yang, Z. Sun, X. Ban, and J. Holguín-Veras. Urban freight delivery stop identification with gps data. *Transportation Research Record: Journal of the Transportation Research Board*, 2411(1): 55–61, 2014. doi: https://doi.org/10.3141/2411-07.

T. Yu and H. Zhu. Hyper-parameter optimization: A review of algorithms and applications. *CoRR*, abs/2003.05689, 2020. URL `https://arxiv.org/abs/2003.05689`.

J. Zhang. Gradient descent based optimization algorithms for deep learning models training. *CoRR*, abs/1903.03614, 2019. URL `http://arxiv.org/abs/1903.03614`.

Y. Zhang and A. Haghani. A gradient boosting method to improve travel time prediction. *Transportation Research Part C: Emerging Technologies*, 58:308–324, 2015. doi: https://doi.org/10.1016/j.trc.2015.02.019. Big Data in Transportation and Traffic Engineering.