Master Thesis

Predicting User's Interest Profile for Public Transportation

Simon Baumann
Roman Prokofyev
Philippe Cudré-Mauroux

August 2016

Contents

0.1 Abstract . . . 2
0.2 Introduction . . . 3
0.3 Related Work . . . 5
  0.3.1 Challenges in Spatial-Temporal Data Analysis Targeting Public Transport . . . 5
  0.3.2 Mining temporal patterns of transport behaviour for predicting future transport usage . . . 5
0.4 Problem description . . . 7
  0.4.1 Sequence Prediction . . . 7
  0.4.2 Proposed Solutions . . . 8
0.5 Experiments and Evaluation . . . 12
  0.5.1 Introduction / Procedure . . . 12
  0.5.2 Data Set / Data Analysis . . . 12
  0.5.3 Naive Approach . . . 15
  0.5.4 Machine Learning Preparation . . . 18
  0.5.5 Machine Learning with WEKA . . . 22
  0.5.6 Experiments . . . 24
  0.5.7 Comparison of Results . . . 37
0.6 Conclusion . . . 40
0.7 Future Work . . . 41

0.1 Abstract

In this thesis we analyze different ways to reliable predict the interest profile of users of public transportation. Having a method where we can successfully predict future behavior enables us to create useful assistances for the user. We can support the way people access information about public transport and proactively suggest trips they are most likely to take in the near future. We also avoid wasting gathered data on the user profiles.

To achieve our goal we applied different machine learning techniques on a real life dataset, gathered anonymously from users of a public transport related Android application. This app collects data about the stations the user has passed on his trip. We want to be able to predict the next station the user is going to as accurately as possible, given information that is available to the system in the moment.

As raw data we have gathered data points that include the user id, the current timestamp and the station id of the closest station. We have analyzed our data set from different perspectives such as the temporal distribution of our data or geographical usage distribution. This analyzes has given us insight into how our data is structured and what limitations are present. We have also created a baseline approach to get a first test of the accuracy, where we achieved 45% correctly predicted entries.

We then applied statistical analysis to further refine and reduce our data set. This was necessary to avoid a distortion of our predictions from users where we don't have enough data or the data doesn't properly represent real-life use cases. We have removed low-profile users and have reduced the amount of stations that we want to be able to predict. We compared multiple machine learning algorithms and analyzed the advantages and disadvantages of each of the algorithms. We executed different experiments with a combination of feature sets with all algorithms and evaluated the results and predictions to draw conclusions to the reliability and the meaningfulness of our approach. We achieved an average accuracy on our predictions of 92%. We further discuss what future work can be done to take advantage of the techniques that we explored in our thesis.

0.2 Introduction

The main goal of our thesis is to accurately predict future behavior of users of our Android application based on historical data we have gathered about each user. We want to be able to support the user in ordinary day to day tasks and to provide assistance for the daily routine. We have gathered geographical and temporal data of users of an Android application that is used for looking up public transportation information. With the data set we should be able to create anonymized user profiles, the precision of which always reflecting the amount of data we have on each user. With this data we try to predict future behavior based on features such as current station, day of week, time and past station(s). We haven't integrated the predictions into the Android application for the end user, but we provide an analysis of the steps that are needed and have evaluated different implementations and their respective prediction accuracy. The implementation we created can be directly used for the integration into the Android application, given some fine tuning of the parameters.

To achieve our main goal we first analyzed our data set, comparing temporal and geographical properties of each user in different combinations. This allowed us to get an understanding of how our data is structured and the possible pitfalls and issues we might encounter. We have been able to include the conclusions from this analysis into account when preparing the data for the machine learning toolkit. The baseline approach we created served as a starting point and a reference for our predictions. We then evaluated and compared different approaches and machine learning algorithms on our data set. We also evaluated different types of use cases and combinations of features available in our data set. We compared the results of these evaluations and tried to find a combination of data preparations, algorithms and evaluations that yielded results successful enough to be able to be used in an actual application.

Machine learning in general has seen an incredible boost of interest in recent years in the research community as well as in industrial applications. Due to the amount of data being generated and gathered across various disciplines new approaches to data analysis had to be discovered. It is no longer feasible and sufficient to manually skim through the data and draw conclusions from these analysis. The process is usually too slow and the amount of data too overwhelming to have a reasonable process of analysis. This issue has led to the rising of new machine learning techniques and related fields which fall under the broader term of artificial intelligence (AI). Many different algorithms and techniques have been developed to account for different problems, data sets and procedures. We take advantage of the research that has gone into machine learning and use a toolkit called WEKA (explained in more detail in chapter

An example use case of the results of our thesis would be a personal assistant that "knows" a users daily routine from historical data and without the user having to explicitly state it. Combined with other data sources such as geographical locations (home, work, gym, train stations, shopping centers, etc.), calendar entries or current public transport timetables the assistant could provide information about when to leave the current place in order to get to the next place that is to be targeted in the users routine.

The part of this thesis in such a use case is to provide the underlying machine learning framework that can be included in such an application. For that we take the available data we have and analyze the necessary steps to get predictions as accurate as possible. The conclusion drawn from this thesis can be integrated in a more comprehensive application that covers different use cases. We will cover this in more detail in the chapter about future work (section 0.7).

0.3 Related Work

0.3.1 Challenges in Spatial-Temporal Data Analysis Targeting Public Transport

Mohammad Sajjad Ghaemi, Bruno Agard, Vahid Partovi Nia, Martin Trépanier
Polytechnique Montrèal, 2015

The first paper we analyzed is from the Polytechnique of Montrèal, published in 2015 and looks at challenges in spatial-temporal data analysis targeting public transport. They have investigated problems and challenges that might arise doing large scale data analysis and sequence prediction on top of publicly available usage data of public transport. They have reviewed these scenarios and suggested solutions to take the different problems into account. However they have not directly worked with large-scale user data to try and validate their scenarios, leaving that as open problems for future research.

Most scenarios that have been analyzed are targeting spatial behavior, the way users use public transport and the trips they take meanwhile. The reasoning behind their analysis is to figure out similar patterns that users have. The assumptions that users with similar usage patterns will also in the future behave similarly can lead to a significant increase in prediction precision. Most usages between users might however not be the exact same sequence of of stations, but a path that bears certain similarities. Scenarios they have listed includes users with the same starting point and ending point, users taking the same buses in opposite directions, users with same directional patterns or same symmetric direction patterns, however not on the exact same stations. Other more complex use cases are the same pattern except for one or two outliers, a shared subsequence of stations or the same resulting distance of travel. They even went further and analyzed similarities of patterns based on a circular grid representation of bus stops or of a pairwise bus stop similarity coefficient. For all the scenarios they have analyzed the possible benefits of using the scenario and possible solutions to incorporate them.

As for temporal data they have presented a distance calculation technique based on the k-means clustering method. In this method the temporal data of a user is encoded as a 0-1 vector. When comparing these n-dimensional vectors between users their similarity can be computed quite easily. This yields a fast and simple way of analyzing similar temporal usage patterns between users.

0.3.2 Mining temporal patterns of transport behaviour for predicting future transport usage

Stefan Foell, Gerd Kortuem; Reza Rawassizadeh; Santi Phithakkithukoon; Marco Veloso and Carlos Bento
Third International Workshop on Pervasive Urban Applications, 8 Sep 2013, Zurich, Switzerland

The second paper we looked at was published in 2013 and tries to predict the future transport usage of users based on temporal patterns. They have done similar predictions as we have in our thesis, however on a coarser level. The goal was to predict whether a user would use public transport on a given day in the future, based on his previous usage patterns.

They have used anonymized data from automated fare collection systems from bus rides in Lisbon to train and test their approach. The highest prediction accuracy they have achieved in their tests is 77%. As a comparison they have included two simple baseline approaches and they have achieved a far higher accuracy using machine learning techniques.

Their data set consists of almost 25 million bus rides taken by over 800'000 travelers over a period of six weeks. At first they have analyzed usage profiles to get an overview over their data set. For that they have plotted the weekly travel profiles, the transport usage periodicities as well as the weekday / weekend travel behavior.

For the actual prediction they have created four different temporal features of their data set. The first is the part of the week, being encoded as either weekday or weekend. The second feature is the day of the week. In addition they have used the travel periodicity of a user, specifying the typical time periods that underlie the user's access to the buses. They also include the travel stationarity, specifying how long a user has been continuously using public transport before the to be predicted day. Using these features they have trained a naive Bayes algorithm with a 80% - 20% split of their data set. They have used a superset of their features for the prediction, leading to lots of combinations in feature sets. The accuracy fluctuates between 0.655 and 0.774.

As a baseline approach they have also compared their predictions to a single ALWAYS and NEVER approach. In the ALWAYS approach they would assume that the user takes the bus on every day and on the NEVER approach that he would never tak a bus. The ALWAYS approach had an accuracy of 52% whereas the NEVER had an average accuracy of 48%. Both baseline approaches are clearly outperformed by the naive Bayes algorithm, no matter the feature set.

As a future work they have presented ways to include more information about the users in the machine learning system, such as the accessed stops of the users as well as the routes and services that are provided. This will allow them to unveil more complex patterns and to give more fine-grained predictions about the usage patterns of the users.

0.4 Problem description

The Android application that we used for gathering our data set is an application that supports users using public transport. Given the current location it will provide you with an up-to-date timetable of the next buses and trains that depart from your closest stations as well as when they leave and where they're going. It uses data freely available from the SBB Open Data Initiative. [1]

The application contains an opt-in service that allows gathering usage data of the application. When a user has opted-in and is using the application, this service will gather information about the closest station and the timestamp and send it to a server where it is permanently stored. The data also includes an anonymized user id to identify the data points to the specific user. This is the data set that we worked with for our thesis. Each entry includes the user id, the station id and the timestamp of when the data point was collected. The reference data didn't contain any structure otherwise, so we started out with a flat file including the full data set. The data set and the analysis we've done on it are described in further detail in section 0.5.2. We have enhanced each raw data point with the previous station the user was at as well as the next station he went to. The next station is what will be used as ground truth in our predictions.

Due to the way the data is gathered in the application and the fact that in a single trip the application will store a data point for each station that is passed (with its respective timestamp) we can conclude that our data points are dependant on each other. An individual data point should not be looked at in isolation because the meaning of the gathered data point can change for our prediction. To successfully predict the next station we need to have information in our historical data about the movements between the stations and not just the stations themselves. Therefore we have modeled our problem as a sequence prediction problem. Sequence prediction is discussed in more detail in the following section.

0.4.1 Sequence Prediction

As machine learning is a very broad field covering lots of different use cases and is based on differing assumptions it is crucial to first analyze the kind of problem that is being tackled by applying machine learning techniques. Without a fond and thorough understanding of the domain and the available data it's nearly impossible to get a useful result and conclusion from experimenting. Just as it is a lot harder for a normal human being to deduce useful information and to learn something given some random, unstructured, unprepared, redundant or even wrong data it is also not possible (at least not yet) for a computer (i.e. a machine learning algorithm) to make sense of such a heap of data. Data needs to be analyzed, prepared, structured and combined before being fed to the algorithm in order to fully unveil its usefulness. If we do not properly prepare the data and find patterns, we are almost certain to run into indescribable issues or results later on in the process. [4]

Many machine learning algorithms are created for independent, identically distributed data.[3] They work under the assumptions that two data points should not correlate and have no explicit influence on each other. In our thesis this is not the case. We explicitly combine data from different data points (such as what was the last station before this one). Therefore we have modeled our problem as a sequence prediction, or sequence learning problem.

Sequence prediction deals with sequential data. A machine learning algorithm that allows for sequential data should not make the assumption that data points are independent, should account for distortion and should also use contextual information whenever available. Popular use cases for algorithms using sequence prediction are time-series predictions (e.g. weather forecasting, stock market predictions, geographical tracking predictions) and sequence labeling (e.g. speech recognition, handwriting recognition, gesture recognition). There are different types of algorithms that fall under this technique, such as different supervised learning classifiers (e.g. Decision Trees, Probabilistic Algorithms, Support Vector Machines, Neural Networks). In our experiments we have included several different algorithms from the field of sequence prediction.

0.4.2 Proposed Solutions

We used multiple different techniques to solve our task and to use in the final evaluation. A decision tree algorithm, a naive Bayes algorithm as well as a multilayer perceptron algorithm (which falls under the Neural Networks category). The theoretical foundation of these algorithms are described here in further detail. The practical way in which we used them are described in Section 0.5.4. As a machine learning toolkit we have used WEKA.

WEKA

In order to not need to reinvent the wheel when it comes to implementing the actual algorithms we have built our solution on top of WEKA, the machine learning toolkit provided by the University of Waikato, New Zealand. WEKA is a collection of machine learning algorithms as well as tools for data processing, clustering, classification, visualization and more. It can also be used to develop new machine learning schemes. WEKA is written in Java and can be used by any JVM-based programming language. Its algorithms can be directly used on a dataset. We only used a subset of the features provided by WEKA. We didn't use any of the visualization tools. Since we also did all the preprocessing and analysis ourselves we didn't need any of the features WEKA provides in that part. What we were interested in is the implementation of the classifiers as well as the actual training, testing and evaluation of the classifiers. As is described later in this thesis we used the provided implementations of the classifiers we propose.

Decision Trees

Decision tree learning in the context of data mining is a technique using a decision tree as a predictive model to map observations about input data. These observations help come to a conclusion about the data's target value. A tree model that accepts a finite set of values is called a "classification tree", in case continuous values are allowed the decision tree is called "regression tree". A decision tree doesn't explicitly represent a decision, however the resulting classification tree can be used as an input for decision making.

In a decision tree model, a leaf represents a class label and a branch represents a conjunction of features leading to the class labels. As can be seen in the example in Figure 1 the interior nodes correspond to the input values, the edges are the value domain and the leafs represent the values of the target variable. With such a modeled decision tree we can easily either visually or computationally evaluate new input data and derive the target values.

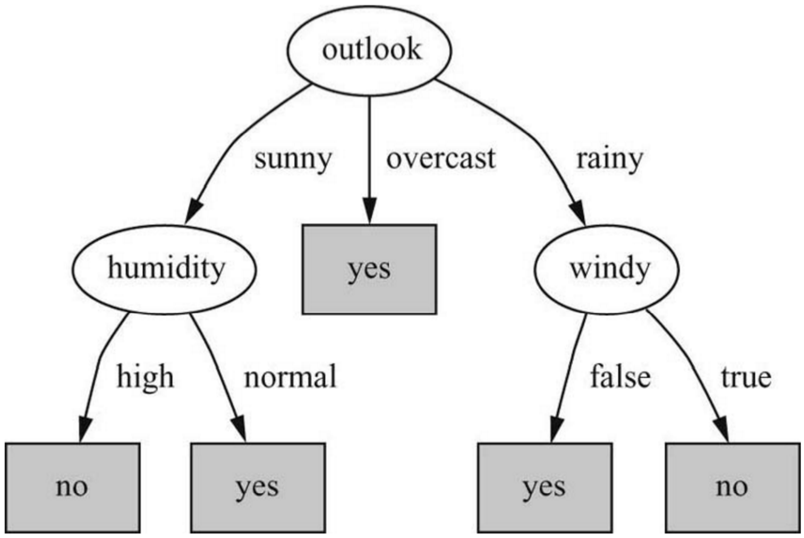


Figure 1: An example of a decision tree [6]

A tree can be trained or learned by splitting the source data set into subsets based on testing the attribute values. In order to get the full model of the decision tree, this process has to be repeated until a subset at a node has only one value or until splitting no longer adds value to the predictions. This process is called recursive partitioning and is the most common strategy to train decision trees.

There are many variants to a simple decision tree that use a combination of multiple decision trees. Examples algorithms are Random Forests, Rotation Forests or Bagging Decision Trees. In addition there are different algorithms that implement decision trees. They usually focus on a different metric that they want to optimize.

Decision Trees have many advantages. They are quite simple to understand and reason about. They also require relatively little data preparation. An additional advantage is that necessary computing resources are quite low, even when using large datasets.

Naive Bayes

Naive Bayes classifiers are simple probabilistic classifiers that are based on Bayes theorem, assuming independence of features. On an abstract level, given a vector representing some features, the naive Bayes will assign a probability to all possible outcomes (also called classes) given this vector. It is a condition probability model. Based on Bayes' theorem, a reliable and computable model can be constructed for all the possibilities that need to be generated. From the naive Bayes probability model we can then construct a classifier. The naive Bayes classifier usually combines the probability model with a decision rule. This rule will define, which hypothesis is to be picked. A common rule is to simply pick the one with the highest probability, which is also called "maximum a posteriori" (MAP) decision rule.

Advantages of naive Bayes classifiers are also their simplicity. If the conditional independence assumptions of the feature are fully given, it will also converge faster than other models. One of the disadvantages that come with conditional independence is that it won't be able to model relations between features properly.

Multilayer Perceptron (Neural Network)

A multilayer perceptron is a special case of an artificial neural network. Artificial neural networks work on a simplified model of how the human brains work. They generate layers of nodes, each having certain input and output values. The nodes (also called neurons) are triggered by an activation function. This function can can take many shapes and be triggered by a combination or a series of inputs of a neuron. Once the activation function has triggered, the neuron will send its output signal throughout the outgoing channels. This might then trigger the next row of activation functions in the next layer of the network, until the output of the network is reached. An example graph of a multilayer perceptron (as a neural network) is shown in figure 2.

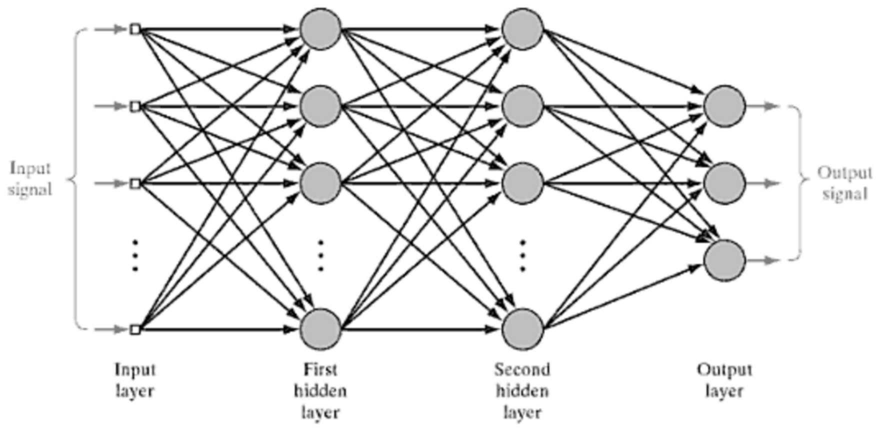


Figure 2: An example of a multilayer perceptron [2]

A multilayer perceptron consists of multiple layers in a forward directed graph, each layer connected only to the next one. Every neuron in the layer has a non-linear activating function to properly model the way neurons in the biological brain are activated. A multilayer perceptron has at least three layers, an input, output and one or more hidden layers. Since it always includes at least one hidden layer it is considered a deep neural network.

A multilayer perceptron is trained using backpropagation. In the beginning of the training stage all weights of all neurons are set to a default value. Based on the errors in the output of each of the training data entry the weights of the neurons are adapted. Once the network is in a stable state and the weights don't change anymore the training can be considered to be completed.

0.5 Experiments and Evaluation

The main goal of our thesis was to compare different state of the art algorithms and to combine machine learning techniques to gain a deeper understanding of what is necessary and what is possible in order to achieve the best results based on an initial data set and certain evaluation targets that one wants to find. In this chapter we explore our procedure from the initial dataset that we work with to the final comparison of the results that we produced.

0.5.1 Introduction / Procedure

At first we did an initial data analysis and created a naive approach that allowed us to gain further insight into our data as a whole and to create a proof-of-concept whether the assumptions we had about our data are justifiable. We then continued to use actual machine learning algorithms implemented into the WEKA toolkit. As part of this we discuss our execution plan, the different features we combined, executed and evaluated and the way we gathered and compared the results.

In order to have data that the algorithms can use directly and to avoid outliers or false cases we combined statistical analysis together with data preparation to convert our raw dataset into a reasonably filtered and correctly formatted set. As discussed in Section 0.4.2 we used a decision tree algorithm, a Naive Bayes algorithm and a multilayer perceptron to train and evaluate our dataset. After improving the process and gathering the results we compared the different execution plans and algorithms and analyzed the impact of different design decisions of our approach.

We have used Groovy to implement all the data analysis, preprocessing and evaluation tasks that we have implemented ourselves. Due to the JVM-based nature of Groovy we could easily and directly interact with WEKA, but we also had all the advantages of rapid prototyping, dynamic types, easy scripting and all the other advantages that Groovy provides on top of Java.

0.5.2 Data Set / Data Analysis

The first step to getting started with machine learning is to know about the data and the underlying domain associated with it. Machine learning is not an oracle where data can be inserted and the computer spits out everything you ever wondered about. Careful analysis and domain knowledge are crucial.

Data Set

Our data was gathered over months from active user of an Android smartphone app. The app (Farplano) gives the user an overview of the current timetable of train and bus stations close to him. The timetable is based on an open data set from SBB (Swiss Railway company). The app contains features such as automatic geolocation, full route information of bus and train lines, arbitrary connection planning and many more. In addition to that it also allowed the user (with an opt-in feature) to track his position and anonymously store the stations with timestamps that he passes. We worked on this gathered dataset from hundreds of users over many months. The raw data contains an entry for every station the user passes. A single data entry contains the following data points:

- Anonymized User ID (a 9-character String)
- Station ID (a 7-character String)
- Timestamp

The information contained in the raw dataset is relatively trivial. In order to be able to successfully predict the future location of a user we combined multiple entries to get further information. As a first step we split up the data set into separate subsets for each user. We also sorted the entries by timestamp to have a chronological view of all the events as they actually happened.

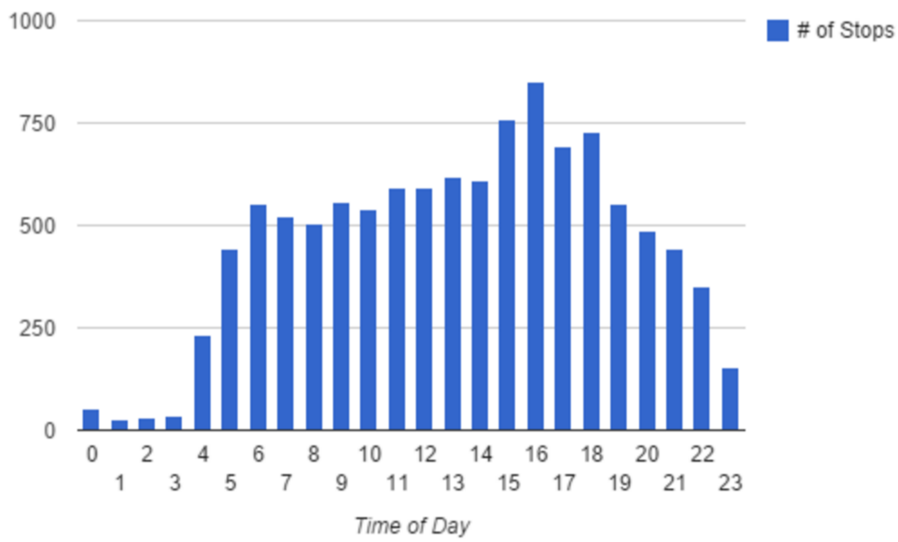
Remark: We have purposefully left out global state and information about usage patterns, mostly due to the added complexity as well as performance reasons. However this might be something that could be analyzed and included in future work.

Data Analysis

To get a basic understanding of what kind of data we have and to be able to reason about our data set we created a number of charts, based on the features we will later use for the machine learning algorithms.

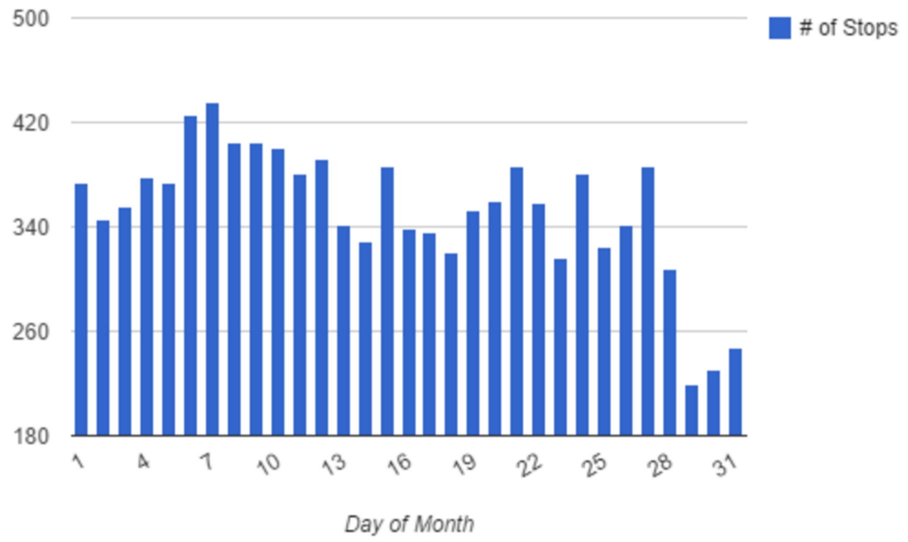
As a first analysis we looked at how the data is distributed by time and day, as can be seen in the following two charts.

Figure 3: Distribution by Time of Day



As we expected we encountered the highest usage during commuting hours, especially in the afternoon. The distribution gets significantly lower as midnight is approached. During the night we have very few gathered data points. The only somewhat surprising point was that the commute hours in the morning

Figure 4: Distribution by Day of Month



didn't produce as high a peak as the hours in the afternoon. However the fact that the data from the app is gathered voluntarily and the app needs to be open to collect data might explain these small inconsistencies.

The distribution by day of month didn't really produce significant insight. The fact that in the end of the month the number of stops are significantly lower is due to fewer months having 31 days. The data set isn't normalized against this and will therefore include such issues. However what this tells us is that the day of month might not be a good indicator. It would be worth exploring the difference between weekday and weekend-day. Since commuting behavior is generally vastly different on weekends as on weekdays this comparison might lead to deeper and more succinct insights.

A vastly more interesting and also challenging conclusion can be drawn from comparing the users with the stations they frequent and how often they stop at stations. As we expected there are a few users that have amassed a lot of data and then there is a long tail of less frequent users. The same statistic also applies to the number of stations and the number of stops that have been gathered for a user. What we already realized here is that it will be very difficult if not impossible to get a good prediction for the users tailing the statistics, simply because there is just not enough data available. In some edge cases where a user really only has 2 or 3 stops that he regularly frequents this might work, otherwise it will just be stabbing in the dark to get a reasonably good prediction. One or two outliers from such a user could possibly mix up the complete prediction process. It seems sensible to cut the dataset into high- and low-frequency users and discard the latter. The exact boundary or whether it will be flexible to a certain degree will have to be tested by trial and error, however if we would not cut the low-frequency user out of our comparison tests it might greatly change our conclusions and the effectiveness of our process.

A similar conclusion can also be drawn by the more averaging figures that we

Figure 5: Number of Stations per User

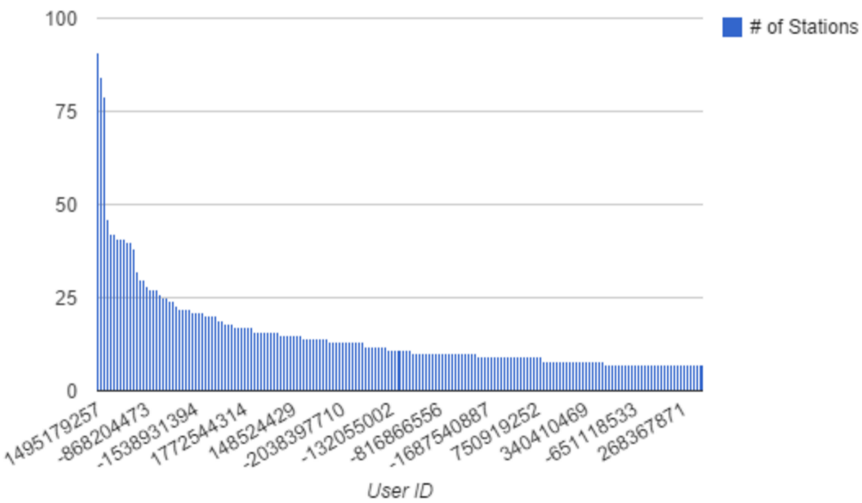
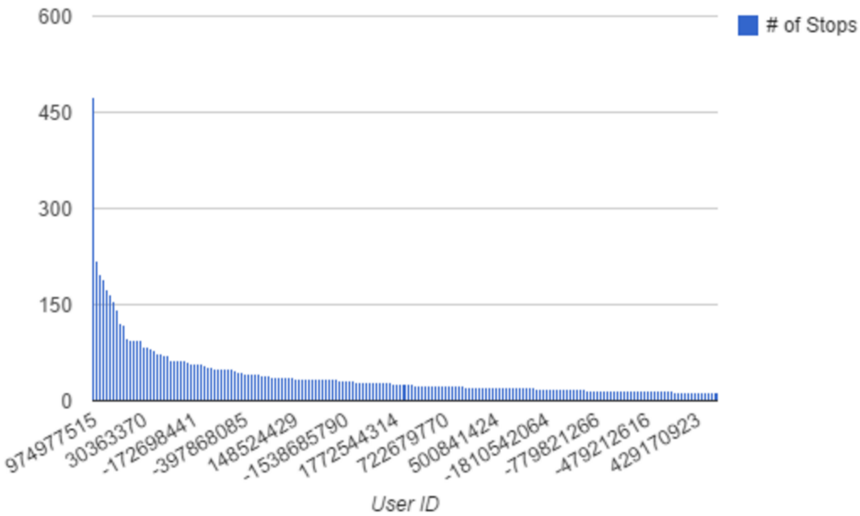


Figure 6: Number of Stops per User



created. It also shows a relatively small set with a lot of data and a long, small tail. Combined with the previous conclusions this strengthened our approach of doing statistical analysis and preparing the data set to remove data points that will skew our result. The types of preparation, analysis and restrictions we've imposed on our data set are described in detail in 0.5.4

0.5.3 Naive Approach

As a first approach we wanted to create a reasonable starting point to work with our data set. For this we created a naive approach in how we first measured

Figure 7: Number of Users per Station

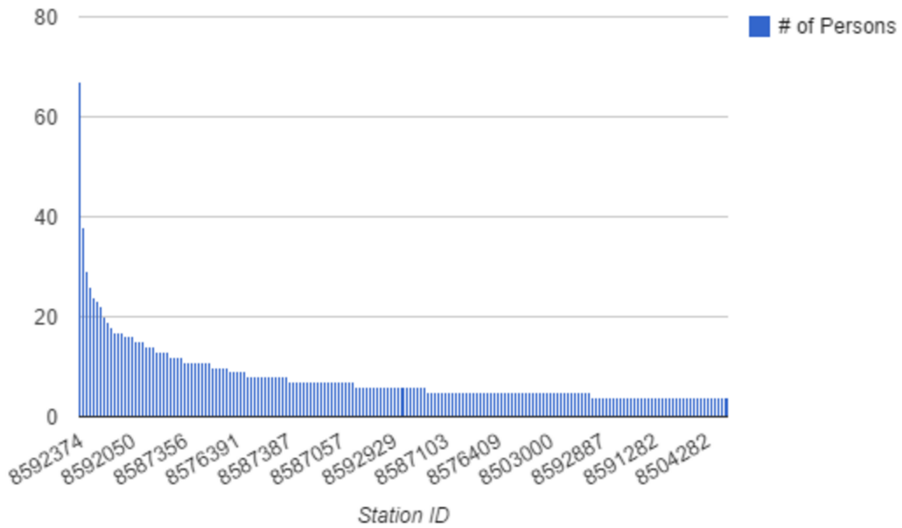
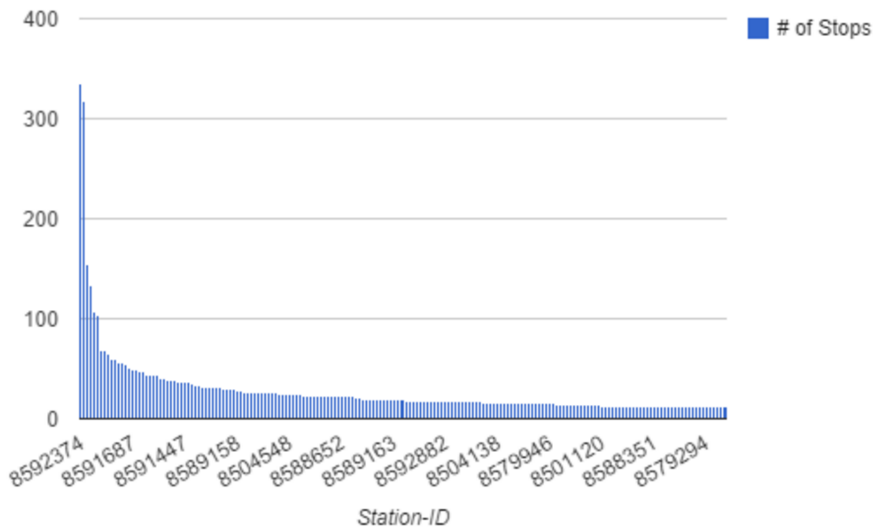


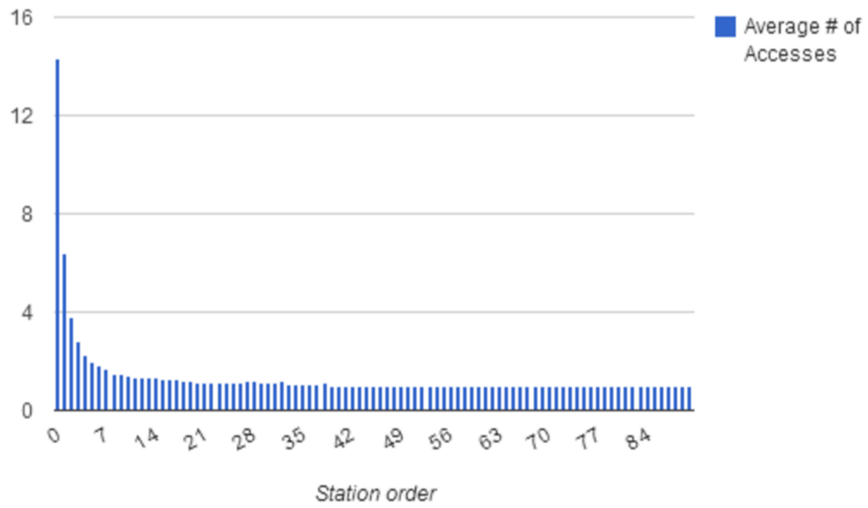
Figure 8: Distribution of Stops per Station



and analyzed our data.

We took our complete data set without doing any statistical analysis or restriction and created a graph for each user. The graph consists of nodes and edges. Each node represent a station and we drew an edge from a node to another node if the destination was the immediate successor of a station in our data set. This resulted in a graph that looked as described in Subsection 0.5.3 and that reflects how the users travel across the network of stations. Our guess was that since a user might generally take similar paths on his travels this would be reflected in the graph and could give us a good first estimate.

Figure 9: Average number of accesses per station by users



For the analysis we then looked at the node that was to be analyzed (assuming this is the node the user would currently be at) and just took the path with the most outgoing edges as our prediction. We then validated this against the ground truth that we had established previously, where we just ordered all the stops at the stations in chronological order. The expected prediction would therefore always be the station that appears next in chronological order for a user.

This naive approach with all its known limitations and issues gave us a correct prediction of 45.7%. We have purposefully chosen not to limit our data set or to put it under certain limitations. This obviously leads to a worse overall prediction and also to some wrong predictions that will be classified as correct, since our ground truth does not necessarily fully reflect the actual reality. However our goal was to start out with a reasonable estimate on the basis of our data set. We didn‘t want to prematurely optimize issues away that wouldn‘t really disturb our prediction models. With that approach we could start out from the very base of our prediction and then analyze the effects that statistical optimizations and a better preparation of our data set have. Thanks to the naive approach that we created we could make sure that what we analyzed with machine learning techniques and the statistical analysis and preparations that we did were actual optimizations and gains in precision.

Web GUI, Visual Representation

Since we were aiming for a reasonable starting point and a general look into our data set we thought it makes sense to also create a visual representation, at least for that first step, before going further into detail. It should allow us to look at individual users and analyze the naive graph that was created for said user. Having such a visual representation helps us in reviewing data for user, seeing at how the data is distributed and can also aid in the analysis of any issues with the data. The resulting graph that was provided by our Web GUI can be seen in Figure 10. We can see all the nodes the user has visited and the relations as well as the amount of relations between each of these nodes. The Web GUI also allowed us to move the nodes around in order to de-clutter the graph, as well as to enable or disable certain nodes. This is especially useful when a user has a huge number of nodes and we only want to look at a certain part of the users graph.

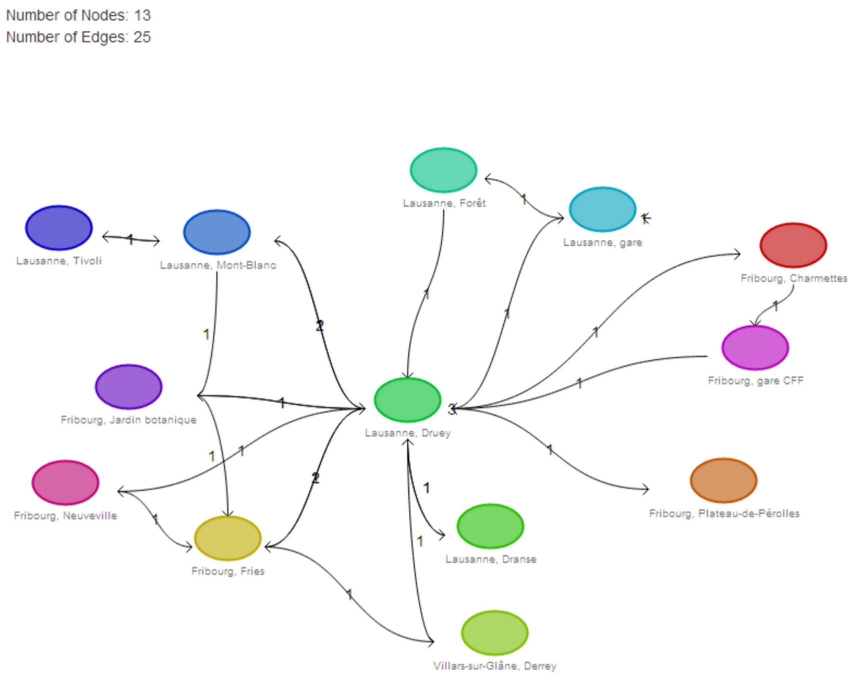


Figure 10: Generated Graph in Web GUI

0.5.4 Machine Learning Preparation

As outlined before applying machine learning techniques to our problem was the main goal of our thesis. After having gathered insights into our dataset by statistical means and by looking at the dataset with a naive approach we started working on machine learning. We laid out our execution plan, defining what and how we want to test the different algorithms on our dataset, what kind of algorithms we'd use and how we evaluate and compare the results. The following sections are the result of carrying out these plans.

Execution Plan

The reason to have a proper execution plan is to have a reproducible way of evaluating the features with different classifiers and reason about the gathered results.

The classifiers that we use are already described in a theoretical manner in section 0.4.2. We have used a decision tree classifier and a naive Bayes classifier both provided from WEKA. As a classifier with back propagation we have used a multilayer perceptron also provided by WEKA.

To test the accuracy of each of the classifiers trained with different features we are using the F1 score. The F1 score considers both the precision and the recall. In information retrieval the precision is defined as "How many selected items are relevant?" whereas the recall is defined as "How many relevant items are selected?". The combination of both precision and recall gives quite a good approximation of the accuracy of a classifier and allows us to easily compare the different results with each other. Mathematically the F1 score is defined as

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{1}$$

The F1 score is also called traditional F-measure or balanced F-score, as it's the harmonic mean of precision and recall and thus weighs both measurements equally.

We are using the precision, recall and F1 score directly to compare the different executions and to analyze their results. However we also split up the result set to compare the precision/recall/F1 for the number of most frequent stations used as well as for different types of users (frequent/non-frequent). With that analysis we should be able to see how the prediction for different usage patterns and types of users are influenced by our classifiers.

The features that we have been able to identify from our data set and that we have worked with are the following:

- Current Station
- Day of Week
- Hour of Day
- Minute of Hour
- Weekday or Weekendday
- Previous Station
- **Next station** (to be predicted)

The current station can directly be deduced from the gathered data. The same can be done thanks to the timestamp for the day of week, hour of day, minute of hour and the weekday/weekend flag. For the previous station we had to do some calculation as well as some assumptions about the data. The way we have prepared the previous station is described in more detail in section 0.5.4.

The next station is the feature that we will be predicting with our algorithms. It is also called our ground truth. We know that our ground truth does not necessarily correspond to the actual reality, but it is the closest assumption we can make with the data and the resources we have at hand. Improving the ground truth would require either the actual implementation of the prediction into the Android app and letting the user give feedback on the prediction or sifting through the data manually and analyzing whether our ground truth makes sense given the user's behavior. Unfortunately both options are out of scope for this thesis, leading us to rely on our somewhat unreliable ground truth.

The issue that could arise from that is that we optimize for a scenario that does not properly reflect reality and that would leave our results useless or require a lot of improvements for an actual real world use case.

In our execution plans we have used different combinations of the feature set above to train and then test the data set. We have always tried to predict the next station feature.

Data Preparation / Statistical Analysis

As part of the process to convert the raw data we have gathered into a WEKA-compatible format we also implemented the findings we gathered from our data analysis, from the naive approach and from the subsequent execution of our plans and the gathering of the results. We constantly improved the results of our predictions by taking the data with the preparation we had done so far, using the machine learning algorithms on the data and then analyzing the results. This process has not been a straightforward path where we knew what and how we needed to implement restrictions and enhancements on our data, but much more a continuous path, leading to small or significant improvements or decreases in precision. Based on the analysis of each step we took we were able to get a good precision, albeit at the cost described in this section. We outline all the transformations, restrictions and enhancements that we made on our data set here.

Data Parsing

The first step in the process was to parse our data set. The data is available as a single large .csv files, each entry consisting of a line such as this:

```
2015-11-26T23:25:46 path=/v1/training?stop_id=8592010 hashacc=595604002
```

Each line starts with the timestamp of the entry, followed by the api path that was made, the id of the station as well as a hash of the user. The relevant data for us is obviously the timestamp, the user hash (which is subsequently directly used as user id) and the station id. The api path is the same for every entry and is irrelevant for our use case. Directly during the import of the data into our model we separated each entry by user and discarded the entries that had an empty user id attached. There is always a timestamp and a station id available. We have also during the import already added the time-based additional fields to our model. These fields include the hour of day, minute of hour, the day of week and whether it's a weekday or weekend day. We also directly set the station id. The previous and next station are added in a later step. A model entry is internally represented by the following fields:

```java
public class ModelEntry {
  private String userId

  private Date timestampStart

  private int hourOfDay
  private int minuteOfHour
  private int dayOfWeek
  private Boolean weekday
```

```
    private String stationId
    private String previousStationId

    // This is the main thing to predict
    private String nextStationId
}
```

Now that we have completed the import of the raw data into our model we have a representation of every user with every entry that was gathered from said user.

Remove Duplicates
The next step is to remove duplicate entries that are present in a user's entry list. Because the Android app responsible for gathering the data and sending it to the server periodically pulls the current location. If a user has to wait at a station for a few minutes it's possible that the app created not only one but multiple entries for this station. In our prediction later we don't want to predict how many times the app would gather data from the same station, but want to predict the actually target destination of the user. That's the reasoning behind removing the duplicate entries. Recognition and removal of duplicates works as follows: We sort all the entries of a user by its timestamp and then look at two consecutive entries. If the time difference of the two entries lies within a certain threshold, we discard the second entry and only keep the first. After some tests we have come to the conclusion that 60 seconds is a reasonable threshold for discarding duplicate entries. Since the app does periodically update its location data we can be relatively sure that within 60 seconds we catch all the cases that are actually just duplicate entries. In case a user has two consecutive entries with e.g. 2 hours time difference, it's more likely that the user is actually starting a new journey at the second entry. Therefore it wouldn't make sense to discard the second entry, especially since each entry is also tied to the specific time and day it was recorded and that information is then also used in our prediction algorithms.

Assign Previous / Next Stations
Once all duplicates are removed from the entries we assign the previous and next station to each entry. As can be seen later in the experiments the previous station is quite beneficial in contributing to overall precision. As a general rule we assigned the station id of the previous data entry of the user as the previous station. However we are doing a split at night at 04.00. At this time a new day in our dataset begins and if the previous data entry was part of the last day we simply set the previous station id of the current entry to "null". This limitation is done in order to not distort the different days too much. We use the same algorithm to set the next station id.

Remove Low Profile Users
Once both the previous and next station have been assigned to an entry we analyze the data for each user to remove all the low profile users. We have realized that users which have barely used the app and gathered very limited data are very hard to predict. We have set the threshold to the amount of data

entries that a user needs to have to 20. It has given us a good compromise to not exclude too many users and in the process only working with very few samples while still maintaining a high precision and good quality of our predictions. It might be an option to fine tune this value, taking into account for example how concise a users data is. For a user that only ever goes to three different stations we might be able to successfully predict his future movements earlier while it is a lot harder to generate e reasonable model for a user that gathers data only sporadically and in lots of different places. However for the data set we have at hand the threshold we chose is a good compromise.

Analyze Station Usage

The next preparation step we take analyzes the usages of the stations for each user. The filtering we do based on this statistical analysis has led to the highest gain in prediction quality, mostly because we have filtered out a lot of noisy data and additional low profile users that generally pull down the overall precision. We first count the number of occurrences of the most frequently used stations of a user. If those most often used stations still have an occurrence count lower than a certain threshold we discard the user. The threshold in our experiments has been set at 10 occurrences, which has shown to be a reasonable trade-off. What we also do is limit the number of stations that we want to predict. It is generally unreasonable to try and predict every possible station that the user could be based on his previous interactions or even based on the full data set. It has also shown that predictions are quite unstable when trying to predict every single possible station given the data that we have. We have therefore limited the number of stations for our prediction feature to ten stations per user. This number has also been gathered after some tests have shown that it gives a good overall prediction. In that step we therefore also remove the next station id from each entry should it not be within the ten stations that are to be predicted. In this case we add the value 'null' as the next station, signaling that we can't make a prediction based the data available in that entry. The value 'null' is also added to the values that the machine learning toolkit can actually predict. In that case it would show the system that none of the most prominent stations could be predicted. If WEKA predicts a 'null' value for a data entry and the ground truth actually does contain 'null' we also count that as a successful prediction, because after all the system also needs to know when it can't yet properly predict something. Obviously with more data and more processing power we might be able to push the number of possible stations higher, but for our evaluations we have left the threshold constant.

After all those operations we have successfully prepared our raw data into a reasonable set that we can be comfortable handing off to WEKA to build the models from our data, evaluate, cross-validate and display the results. This process as well as the results are discussed in the following section.

0.5.5 Machine Learning with WEKA

Before we present the results we want to outline the way we pass data to WEKA and the computation steps that WEKA internally does.

The data that we can pass to WEKA need to be represented by Attributes. Each feature that we defined is matched by its representative attribute. The instances of the data that are passed to WEKA should all contain allowed values for the attributes. An attribute can have one of the following types (from [5]):

- numeric: represents a number (floating-point)
- nominal: represents a fixed set of values
- string: represents a dynamic set of values
- date: represents a date
- relational: represents a relation between multiple instances

We have only used nominal attributes since the data set we had always consisted of a value from a fixed set. Since we haven't used the full date directly but have split it up into day, hour, minute, etc. we have also used nominal values for these. The set of values for current, previous and next station for a user is every value that each of the features consisted of (including 'null' for previous and next station).

For each entry available for a user we have created an Instance. A WEKA Instance consist of a name (the user id in our case), every possible attribute and the actual value for each attribute. Each instance also can also define a class index that will be used to represent the class for prediction purposes. We have always set the class index to the attribute that represents our next station feature.

The instances are passed to a classifier from WEKA. A classifier is the actual algorithm that builds the appropriate model for our instances. There are many classifiers that are provided by WEKA directly, such as J48 (a decision-tree based implementation), NaiveBayes or MultilayerPerceptron. There are also many open source libraries that extend the Classifier interface from WEKA that can be directly integrated into the WEKA toolkit. Classifiers can then build the model from the instances that are passed. It's also possible to incrementally create or update the model of a classifier. Since we only did offline training and testing this was not needed in our case. However should our system ever be used with real world data we would most certainly need to use incremental model building as we would not want to recreate our full model each time our data set updates.

Once the model is created we use an Evaluation to evaluate the model. There are different ways to train and test a model. One way is to generate a separate training and testing set (usually splitting the original data set by 2 to 1). Another way is to use (k-fold) cross-validation. In cross validation, the original data set is split into equally big subsets. One of those subsets is retained for testing, he other k-1 subsets are used for training. Usually this process is repeated multiple times to erase possible issues with the randomness of the subset generation. We have used 10 times 10-fold cross-validation (where enough data was present) to train and test our instances with our classifiers.

Once the classifiers has been trained and tested, WEKA provides an abundance of information about its predictions. We used this information to put together charts and statistical results for each of our classifier given each of the different feature sets we prepared for it.

0.5.6 Experiments

We have conducted three different experiments on our data set, each with different algorithms and with multiple feature sets. The algorithms are always Decision Tree, Naive Bayes and Multilayer Perceptron. On the last experiment we have omitted the Multilayer Perceptron algorithm due to severe performance issues.

For each experiment and each algorithm we have used three different feature sets to train and test the algorithm:

Stations	**Current Station & Time-Based**	**All Features**
Current Station	Current Station	Current Station
Previous Station	Hour of Day	Hour of Day
	Minute of Hour	Minute of Hour
	Day of Week	Day of Week
	Weekday	Weekday
		Previous Station

Comparison of all Users

For our first experiment we take our fully prepared data set, run each algorithm with each feature set once per user and collect the results of the evaluation. We then sort the results of the users by their F-Values for the line chart in order to get a good representation of how precision and recall correlates and the F-Measure deteriorates in our data set. The Boxplot graph gives a good statistical overview of the distribution and the highs and lows in our result set. As it is generally helpful to have multiple representations of data we have in most cases included both the line chart and boxplot graph.

Decision Trees

Figure 11: Stations Features for Decision Tree

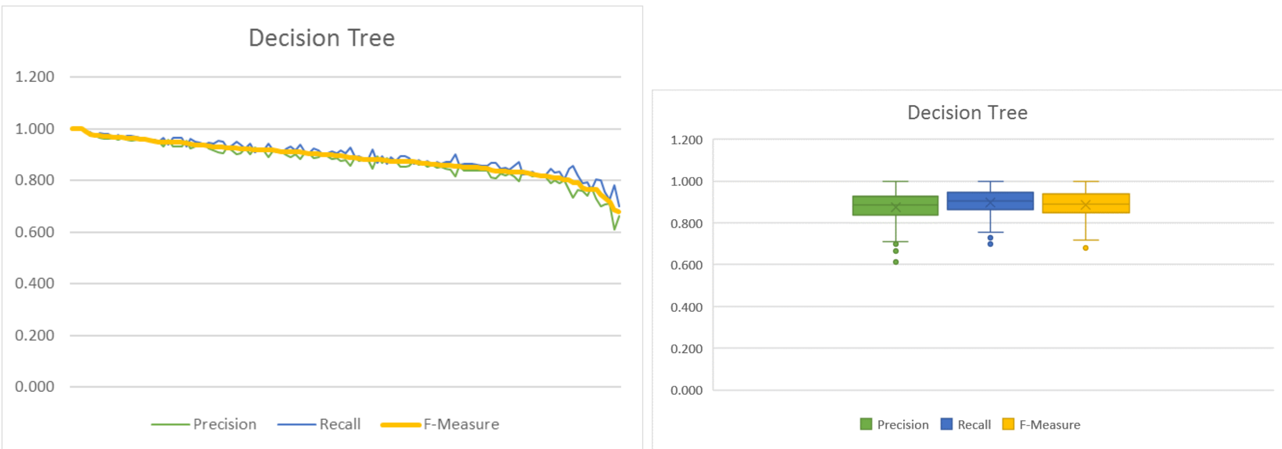


Figure 12: Current Station and Time-Based Features for Decision Tree

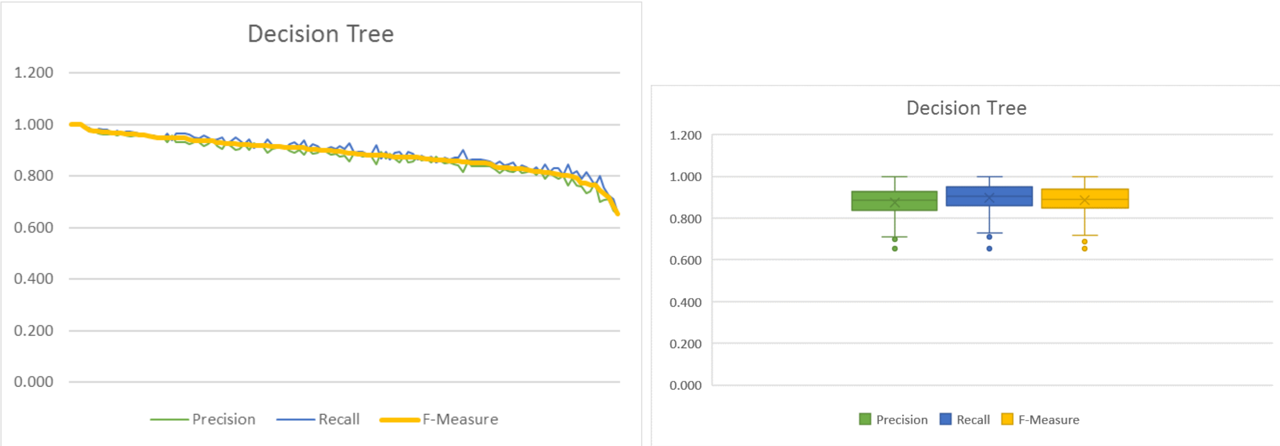


Figure 13: All Features for Decision Tree

Naive Bayes

Figure 14: Stations Features for Naive Bayes

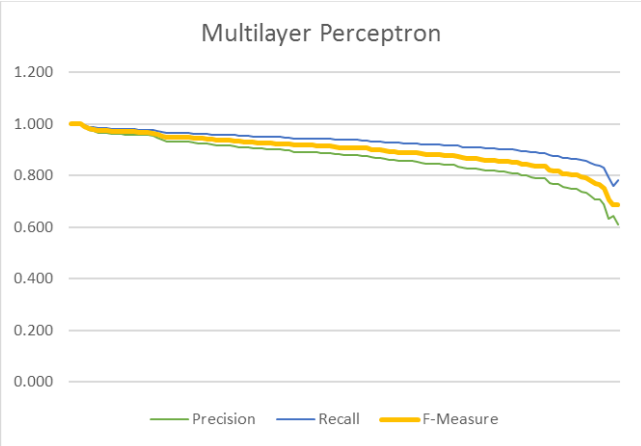
Figure 15: Current Station and Time-Based Features for Naive Bayes

Figure 16: All Features for Naive Bayes

Multilayer Perceptron

Figure 17: Stations Features for Multilayer Perceptron

Figure 18: Current Station and Time-Based Features for Multilayer Perceptron



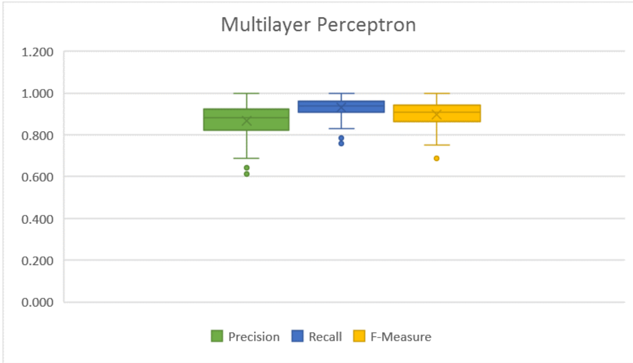


Figure 19: All Features for Multilayer Perceptron

Comparison of frequent versus non-frequent Users

In this experiment we have let the algorithm run through our data set as before, however before creating the graph for our evaluation we have split the result set into high- and low-frequency users. The split is done by 50% of our data set. We simply count the number of data entries a user has to see whether that user belongs into the high- or low-frequency belt. In the graphs below the left-hand side of the graph represent the high-frequency users and the right-hand side the low-frequency users. Each half is then ordered by their respective F-Values just as before.

Decision Trees

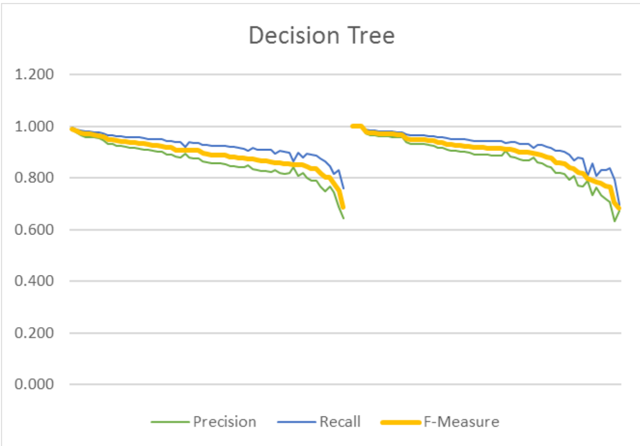


Figure 20: Stations Features for Decision Tree

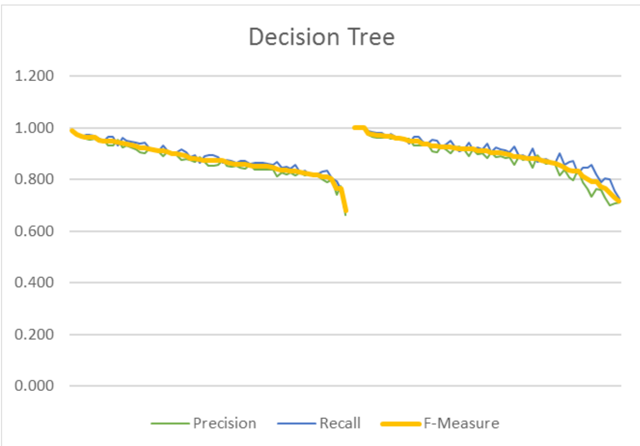


Figure 21: Current Station and Time-Based Features for Decision Tree

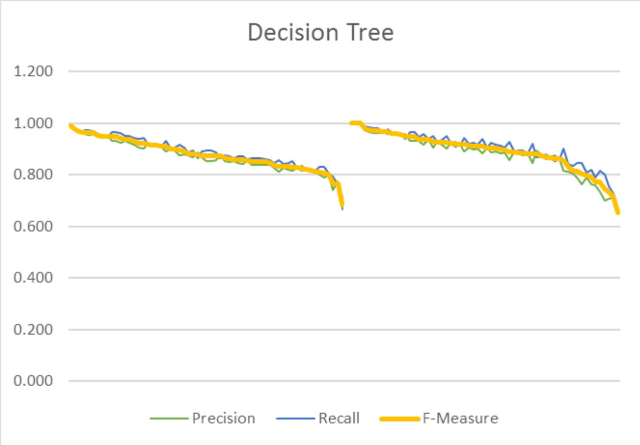


Figure 22: All Features for Decision Tree

Naive Bayes

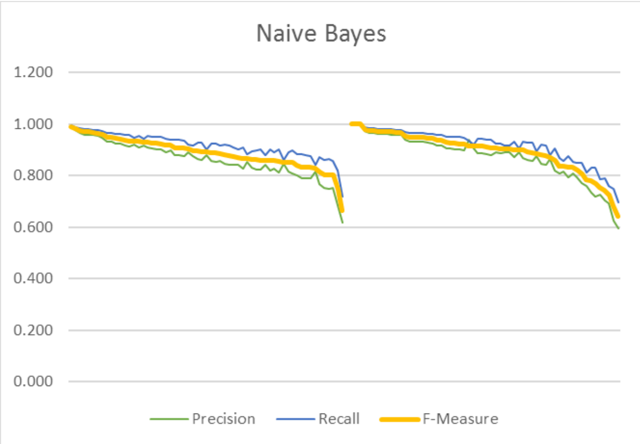


Figure 23: Stations Features for Naive Bayes

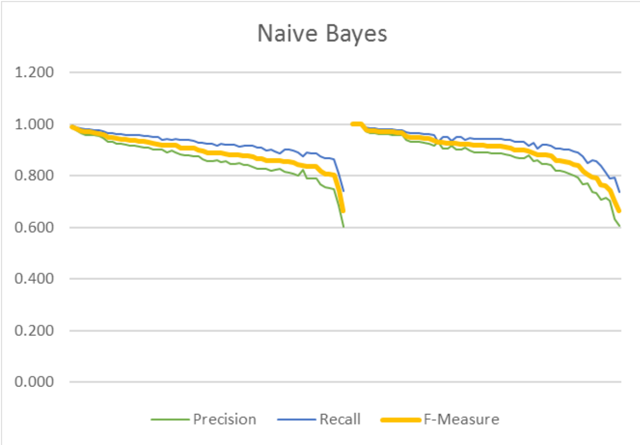


Figure 24: Current Station and Time-Based Features for Naive Bayes

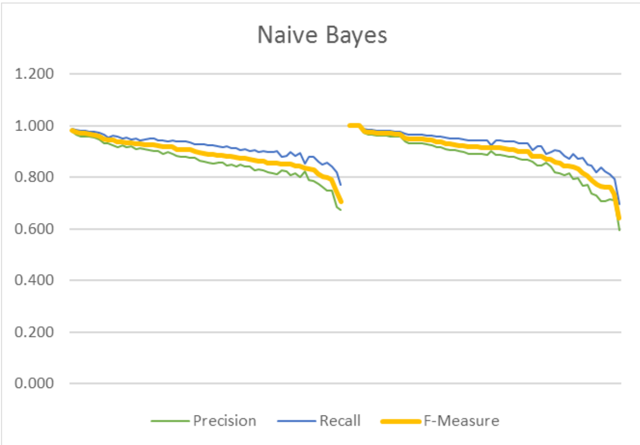


Figure 25: All Features for Naive Bayes

Multilayer Perceptron

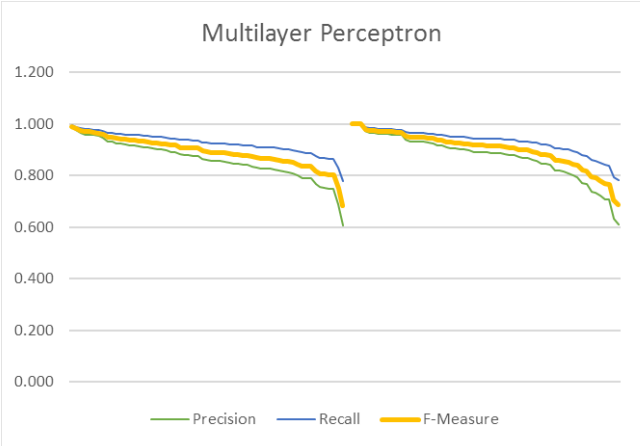


Figure 26: Stations Features for Multilayer Perceptron

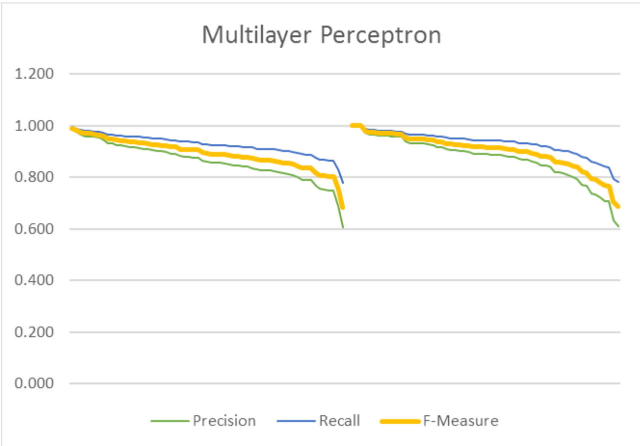


Figure 27: Current Station and Time-Based Features for Multilayer Perceptron

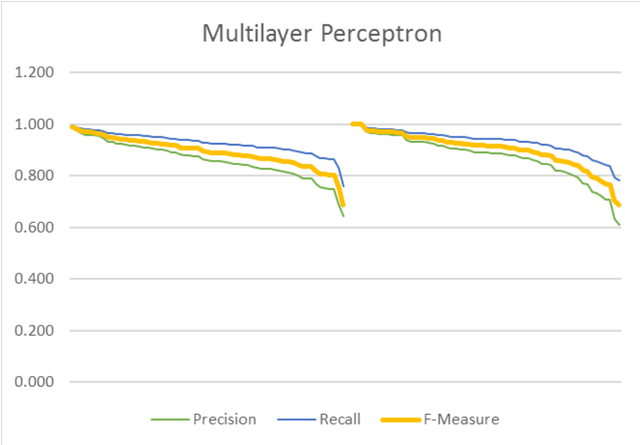


Figure 28: All Features for Multilayer Perceptron

Comparison of all Users with higher number of Stations

For our last experiment we have played a bit with the parameters of the number of stations we include in our prediction and the minimum amount of occurrences the stations have to have. The way its used is described in more detail in subsection 0.5.4. We ran the simulation multiple times with higher and lower thresholds to see what changes. We have taken two thresholds that represent the tests in a good way. In this experiment the maximum amount of stations is set to 15 whereas the necessary amount of occurrences is reduced to five. This has led to a steep increase in users that are actually processed (almost double). Due to the higher user count and the increased complexity, we have not included the Multilayer Perceptron Algorithm in this experiment. It was multiple orders of magnitudes slower and didn't yield significantly different results.

Decision Trees

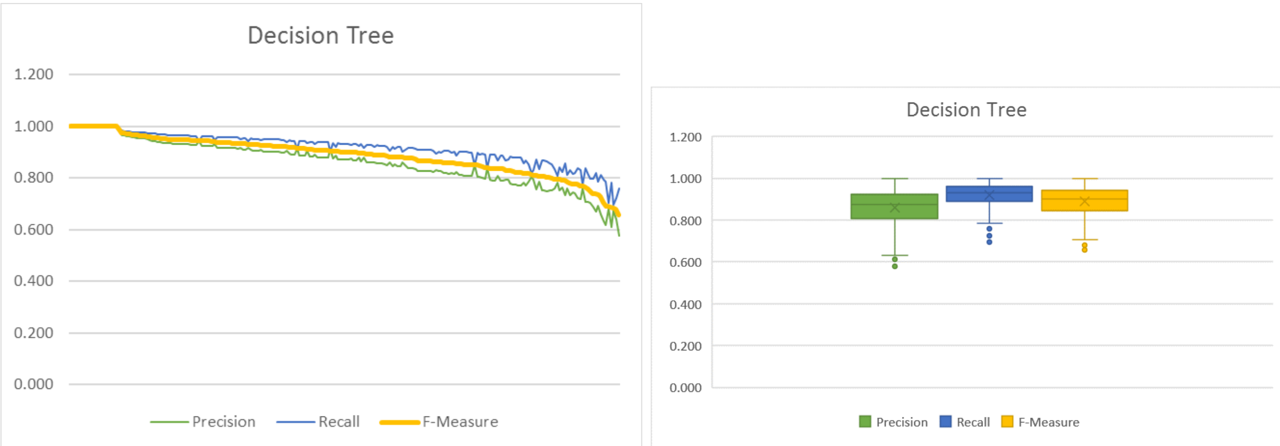


Figure 29: Stations Features for Decision Tree

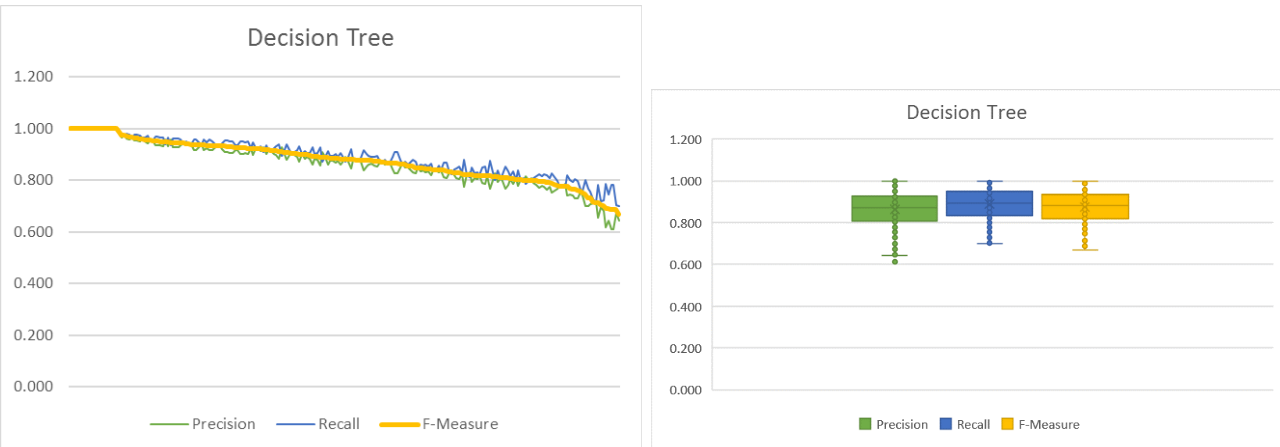


Figure 30: Current Station and Time-Based Features for Decision Tree

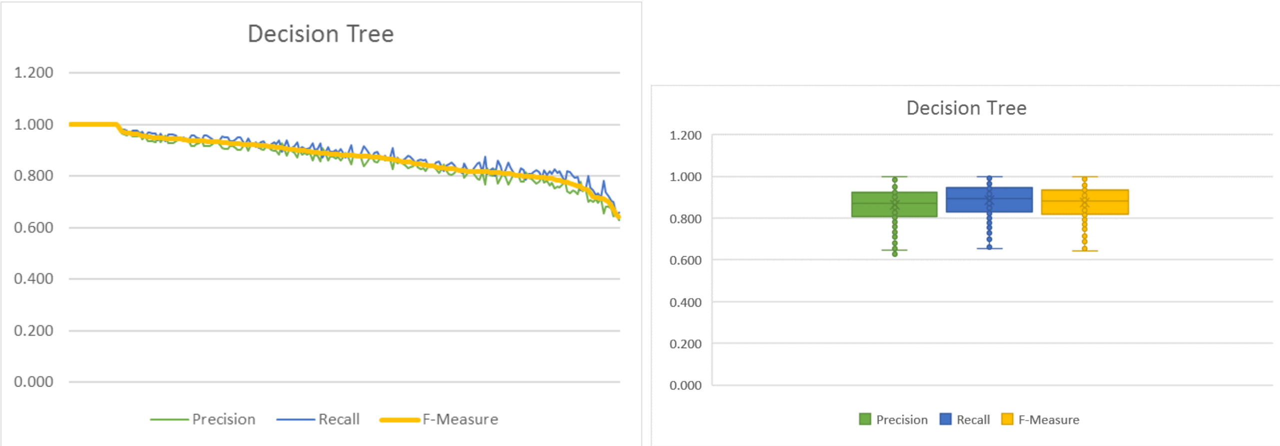


Figure 31: All Features for Decision Tree

Naive Bayes

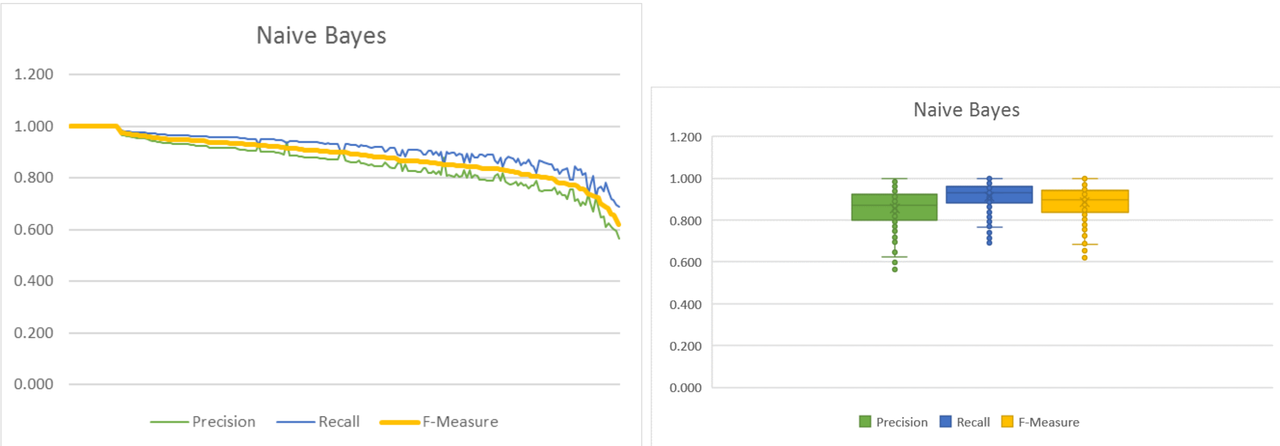


Figure 32: Stations Features for Naive Bayes

Figure 33: Current Station and Time-Based Features for Naive Bayes

Figure 34: All Features for Naive Bayes

0.5.7 Comparison of Results

Comparison of different algorithms

In general it can be said that the selection of the machine learning algorithm hasn't heavily influenced the final results. The F-Measure is quite similar in all of the algorithms, given the same feature set. What is quite different in the algorithms is the consistency of how precision and recall correlate. In our Multilayer Perceptron experiments the precision and recall lines are a lot smoother, meaning that they deteriorate in combination and are quite stable across all users. Recall is usually higher than precision, but the difference between them is stable. This was also apparent between the different feature set. For Naive Bayes and to an even larger extent the decision tree algorithm the curves for precision and recall have lots of small mountains and valleys, especially when looking at the results with lower precision. This leads to the conclusion that the results are more dependent on the specific data for the Naive Bayes and Decision Tree algorithms and we might have some over- or underfitting for these algorithms. The overall precision hasn't suffered too much, but the Multilayer Perceptron algorithm is certainly the most stable of all.

For all algorithms we have experienced a relative straight line up unto approximately 85-90% of the users where we see a quite sharp decline. For those users it might be interesting to check with a different algorithm and see whether the results are similarly low. If the precision doesn't get better we would most likely need to filter them out until more and especially more precise data has been gathered. Because a precision and recall of around 0.7 generally doesn't give us a big benefit. However for about 50% of our users in all algorithms we have a F-Measure of 0.9, which could probably be used in a production system. Combined with some feedback from the actual user that then explicitly validate our ground truth we can most likely eliminate or at least reduce our false predictions for our already high precision use cases.

Comparison of different feature sets

The overall difference between the different feature sets is also quite small. Especially when looking at the F-Measure curve, they show quite similar results, including the difference in algorithms. An interesting observation is that with the addition of time-based features the precision and recall for the decision tree algorithm are a lot closer. The overall F-Measure is comparable, but the difference in precision and recall is smaller. The same change also includes quite a bit of stability for the Naive Bayes algorithm, smoothing out the precision and recall curve. The Multilayer Perceptron didn't show much difference, only some small inconsistencies at the tail of the curve. However since the Multilayer Perceptron is already quite stable the addition of time-based features hasn't helped much.

The distribution of values (from our boxplot charts) across the feature sets are also quite similar. The precision and F-Measure are mostly similarly distributed for each algorithm, the recall is varying a bit. Mostly it is the extrem values that vary, albeit only the lower ones, as the upper extremes are always 1.0. For the recall of the Decision Tree and Naive Bayes however we also see slight changes in the overall distribution. These effects cannot really be attested for the Multilayer Perceptron.

Comparison of different experiments

We can mostly compare the first and third experiment and separately see how the results of the second experiment differ and how they could benefit our overall prediction model. The first experiment is quite conservative in the amount of data that are actually used in our machine learning algorithm. This is based on the assumptions that the algorithms can't cover all the special cases of our data and that the overall precision heavily benefits from our preprocessing and filtering. The third experiment also build upon some parts of this preprocessing and filtering, it is however far less restrictive, thus allowing a lot more and also a lot more low-frequency user to be included in our machine learning process. The results between those two experiments have shown what was to be expected. We have a much higher spread in our third experiment between the highest and the lowest F-Measures. The overall precision and recall are also a bit lower, however not by a huge margin. One of the interesting conclusion is that our third experiment had a lot more perfect predictions (precision = recall = F-Measure = 1.0). This can most likely be attributed to the fact that some low profile users are really easy to predict, if they have very few stations that they have used. Some of these users have already been filtered out in our first experiment. Whether these predictions would also hold up in reality or whether they can just be attributed to our data set is another question that can't be answered by our models.

The issue with using the third experiment is that since the prediction for many more users fall below a usable threshold a lot more data needs to be discarded after our somewhat expensive machine learning calculations. Already filtering theses users out before we send the data through the machine learning system saves time and resources, all the while not generating a hugely negative impact on many of the users. Therefore it might make more sense to go with the first experiment or to use a combination of both, allowing the parameters to be set a bit more flexible for different users.

The second experiment is split in itself and compares high-frequency with low-frequency users. In the test data that we had the split was around 85 actually usable data entries (without all the removed entries from our preprocessing steps). Interestingly enough the curves looked very similar for both high- and low-frequency users. Both had a very sharp decline in the tail of the curve and otherwise quite consistent results. The decline is a bit more pronounced in the high-frequency users. This might help when trying to cut out the lowest X% of the user set. But the overall prediction, recall and F-Measure values are very similar. It seems that when the users reach a certain amount of data entries it's not necessarily the pure amount of data that actually helps with the prediction, but a combination of other factors, such as the complexity of the data, the amount of different stations and the temporal distribution of the values.

0.6 Conclusion

One of the most important goals we wanted to achieve was to find a way to reliable predict the public transport behavior of our users. Given the limitations we placed upon the data necessary for the task we have achieved a respectable result. We have been able to run our machine learning algorithms for 150 - 250 users, depending on the use case, and feel comfortable of using the predicted results of at least 100 - 150 users. While this is a good start, the amount of users that could benefit of our system is still quite small. We have run our predictions for 5-10% of our user base. This is mostly influenced by the missing amount of data. Over 75% of our users have less than ten distinct data points gathered. This is in no way enough data to successfully make a prediction and be convinced about it enough to present it to our user. For the vast majority of users the data gathering would need to be optimized before we can think about including them into our machine learning system.

However for the users that have enough good quality data and that can be evaluated we have seen a good overall prediction. For most algorithm and feature set combination we have a median F-Measure of over 0.9. If we'd cut off the lowest 10% of our predicted users for which the reliability of the predictions is too low, we'd get an even hight median, close to 0.95. This is a very good starting point and with direct feedback from the user as to whether our predictions match the ground truth it could be improved even faster.

We have seen that while the Multilayer Perceptron was by far the slowest of all algorithms, it also performed best. It had a slightly better overall prediction and was also more stable than the decision tree algorithm and the naive Bayes. Precision and recall in general was very similar, but if computing costs are no issue then the multilayer perceptron outperforms the other algorithms.

We have successfully been able to achieve the goal we set out to, creating a useful and implementable way of predicting user behavior based on historical data of public transport usage of these users.

0.7 Future Work

Even though we have successfully completed our experiments and have, under certain restrictions to our data, gathered a respectable precision with our predictions, there are still things left to be improved. In this final section we list the tasks that could and / or should be taken up following this thesis.

As a logical next step the approaches that we have implemented can be included in the actual Android application where the data is gathered. If carefully and usefully integrated into the application that the user actually benefits but doesn‘t consider it as too intruding or too imprecise in the predictions this could lead to a virtuous cycle. The more use the user has the more he uses the feature, in turn providing our system with more data which generally leads to a better prediction for the user. The introduction of the feature would need to carefully monitored, so that it is only active for user with enough gathered data that the system is confident enough of providing a good estimate. Otherwise the prediction might be too shaky, leaving the user with little benefit and generally loosing prospective users in the process. It might be necessary to do some A/B tests to figure out what the target precision for a prediction needs to be in order to display it to the end user. Depending on how much or how often the user likes to see recommendations the precision could be more or less important and the behavior of the application can be adapted.

In combination with creating an actual integration a more flexible approach to the statistical analysis, the data preparation as well as the options of the machine learning algorithms might be taken. After our analysis phase and some trials on the data we have imposed fixed limits on what conditions our data needs to have in order to be relevant for our analysis. Depending on how the user actually behaves it might make sense to either lift or further limit certain restrictions, based on the complexity that the users data profile inhibits. The overall precision of the system might not be improved too much, however for certain edge cases this could lead to significant gains or could successfully delay the integration of the system until enough data and information is gathered to be precise enough.

Another type of prediction that we could try to establish is the current position of the user. We have in our whole thesis only ever predicted where the user might go next, however it could also be interesting to get a best guess for the current location of the user, based on time, day, previous usages of the app, and so on. This might help in getting a faster coarse location, faster than GPS and cell signal triangulation. If the predictions would be precise it could easily lead to a great contribution for the Android application.

On a technical level there are multiple options to go forward. One would be to include a form of global state for the prediction. We have only ever looked at a users data profile in isolation. It might be interesting to evaluate certain patterns that are valid for all users and take these into account when creating a prediction for a single user. This might especially be helpful if the system encounters a case where previous data from the user doesn‘t yield a probable prediction. Since humans tend to be herd animals and follow similar patterns this could be a helpful contribution.

Another technical option might be to switch the machine learning framework for something more recent. A good example to evaluate would be Google‘s TensorFlow, an open source framework based on Python that is founded on the years of research that Google has put into machine learning. TensorFlow is based on neural networks and can be scaled from a smartphone to a large-scale datacenter. It might be very interesting to see what advantages can be gained using an absolute state-of-the-art framework that is already being used in applications by billions of users. The exact effect it would have on our problem set is hard to predict, but it could very well lead to an improved predictions while making less restrictions on the data.

Bibliography

[1] SBB AG. Sbb open data, 2016.

[2] Abdelrahman Elogeel. Multilayer perceptron, 2016.

[3] Alex Graves. Machine learning, sequence learning introduction. University Lecture, 2009.

[4] Mark A.Hall Ian H. Witten, Eibe Frank. *Data Mining - Practical Machine Learning Tools and Techniques*. Elsevier Inc., 3rd edition, 2011.

[5] University of Waikato. Java documentation of weka, 2016.

[6] Wikipedia. Decision tree learning, 2016.