Master Thesis

Predicting User's Interest Profile for Public Transportation

Simon Baumann

February 2016

1 Abstract

In this thesis we apply different techniques of machine learning to a real life dataset, gathered anonymously from users of an Android app that collects data about the stations the user has passed. Our goal is to best predict the next station that the user is going to, based on information that would be available to us in the moment. Such information could include time, day, current location, past locations.

We show ways to analyze and optimize the raw data, prepare and filter it in a way that yields the best results for further analysis. We compare different machine learning algorithms on our dataset and discuss the advantages and disadvantages that we encountered with each algorithm. We then present the experiments that we did on our dataset and evaluate the results and predictions from the algorithms to draw conclusions to the reliability and the meaningfulness of our approaches. We further discuss what future work should or could be done to take advantage of the techniques that we explored in our thesis.

2 Introduction

Machine Learning 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 have to be discovered. It is no longer feasible and sufficient to manually skim through the data and draw conclusions from these analysis. The process would usually be 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.

The main goal for our thesis is to evaluate and compare different approaches and different Machine Learning algorithms on an existing data sets. We also evaluate different types of use cases and different combinations of features that are available in our data sets. We compare the results of these evaluations and try to find a combination of data preparations, algorithms and evaluations that yield results successful enough to be able to be used in an actual application.

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 can create anonymized user profiles, the precision of which always reflecting the amount of data we have on each user. Based on the historical information we have about each user we try to predict the future behavior of the user, based on features such as current station, day of the week, time, past station(s). Our goal is to accurately predict the next station the user will go to and therefore be able to support the user in ordinary day to day tasks.

An example use case would be to create a personal assistant that "knows" from historical data and without the user having to explicitly state it what the daily routine of the person is and to assist it in recurring behavior. Combined with other data sources such as geographical locations (home, work, gym, train stations, shopping centers, etc.) 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 could be included in such an application. For that we take the available data we have and analyze the necessary steps to get predictions as reliable as possible. The conclusion drawn from this thesis could then be integrated in a broader application that covers different use cases. We will cover this in more detail in the chapter about future work (section 7).

3 Related Work

Here goes the related work.

4 Sequence Prediction

As machine learning is a very broad field covering tons of different applications, use cases and is based on differing assumptions it is crucial to first analyze what kind of problem is supposed to be tackled by applying machine learning techniques to it. Without a fond and thorough understanding of the domain and the available data it's almost impossible to get a useful result and conclusion from doing experiment. 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 simply make sense of a heap of data. Data needs to be analyzed, prepared, structured and combined before being fed to the algorithm in order for it to fully unveil its usefulness. If we do not properly prepare the data we are almost certain to run into indescribable issues or results later on in the process.

Many machine learning algorithms are created for independent, identically distributed data. 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. Why we used sequence description is described in section 4.1.

Sequence prediction deals with sequential data. A machine learning algorithm that allows for sequential data should not make the assumption that the 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 out of the field of sequence prediction.

4.1 Task Description

We model our problem as a sequence prediction problem due to the fact that we have to work with heavily dependent individual data points. Due to the way our data is structured we need to assume that different data points rely on each other. As will be explained in more detail in section 5.2 we gather a number of values for every data point. Among these values are the current station (mapped as an id), the anonymized user id and a timestamp. While preparing our data we enhance the data points with references to the previous and the next station.

4.2 Proposed Solutions

We used multiple different techniques to solve our task and to use in the final evaluation. Decision Trees and Random Forests, a Naive Bayes algorithm as well as Hidden Markov Models (which fall 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 5.4

4.2.1 WEKA

4.2.2 Decision Trees

Decision tree learning in the context of data mining is a technique using a decision tree as a predictive model to map observation about input data to 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.

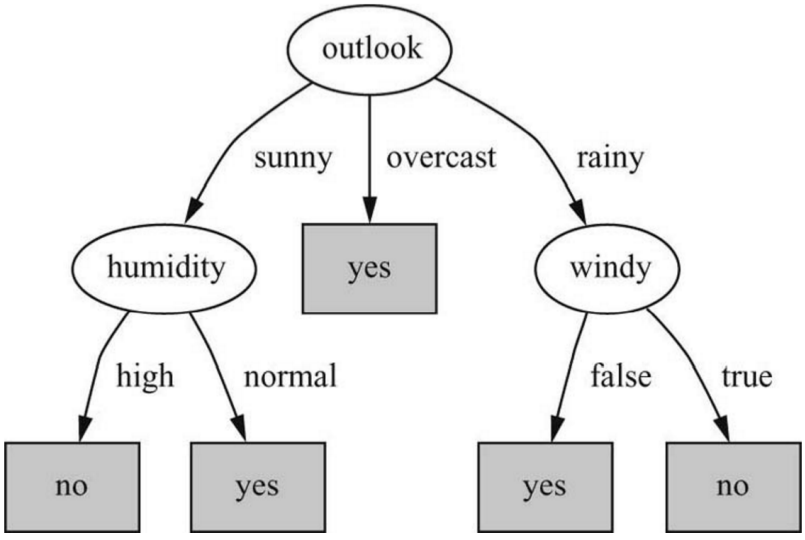


Figure 1: An example of a decision tree

[2]

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 it is the most common strategy to train decision trees.

4.2.3 Naive Bayes

Here goes the theoretical description of Naive Bayes

4.2.4 HMM / Neural Networks / Multilayer Perceptron

Here goes the theoretical description of Hidden Markov Models and Neural Networks

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.

5.1 Introduction / Procedure

At first we did an initial data analysis and created a naive approach that as a way to gain further insight into our data as a whole as well as to create a proof-of-concept whether the assumptions we had about our data are justifiable. We then continue to use actual machine learning algorithms implemented into the WEKA toolkit that is created and maintained as an open source project by the University of New Zealand. As part of this we discuss our execution plan, the different features that we want to combine, execute and evaluate and the way we will gather the results and compare them.

In order to have data that the algorithms can use directly and to avoid outliers or false cases we combine statistical analysis together with data preparation to convert our raw dataset into a reasonably filtered and correctly formatted set. As discussed in Section 4.2 we used a Decision Tree, a Naive Bayes algorithm and a Hidden Markov Model to train and evaluate our dataset. After improving the process and gathering the results we compare the different execution plans and algorithms and analyze the impact of different design decisions of our approach.

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 all the data can be inserted and it spits out everything you ever wondered about. Careful analysis and a knowledge of the domain of the data are inexcusable.

5.2.1 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 thus contains an entry for every station that 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 thus relatively trivial. In order to be able to successfully predict the future location of a user we needed to combine multiple entries to get further information. As a first step we split up the data set to separate sets for each user. We also sorted the entries by timestamp, so that we 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.

5.2.2 Data Analysis

To get a basic understanding of what kind of data that we have and to be able to reason about our data set we created a number of charts, based on the features that 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.

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 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 could 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

Figure 2: Distribution by Time of Day

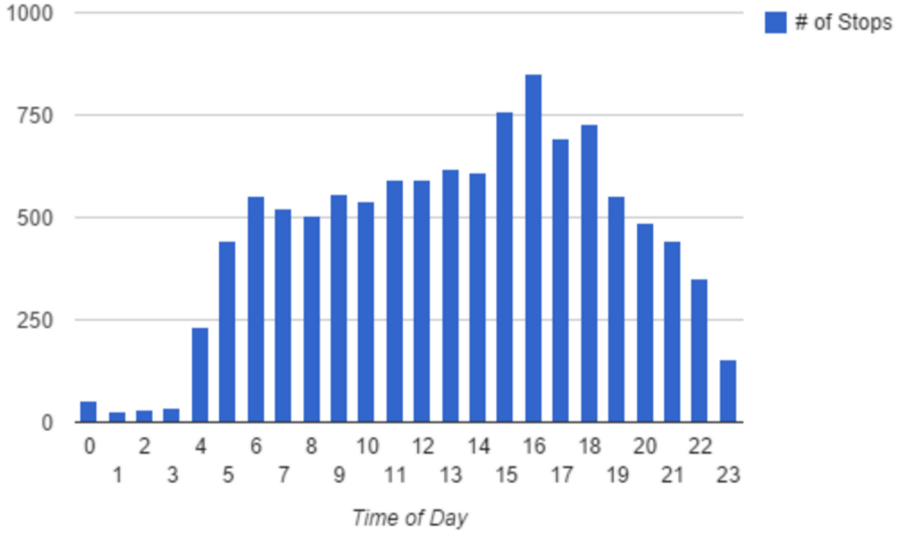
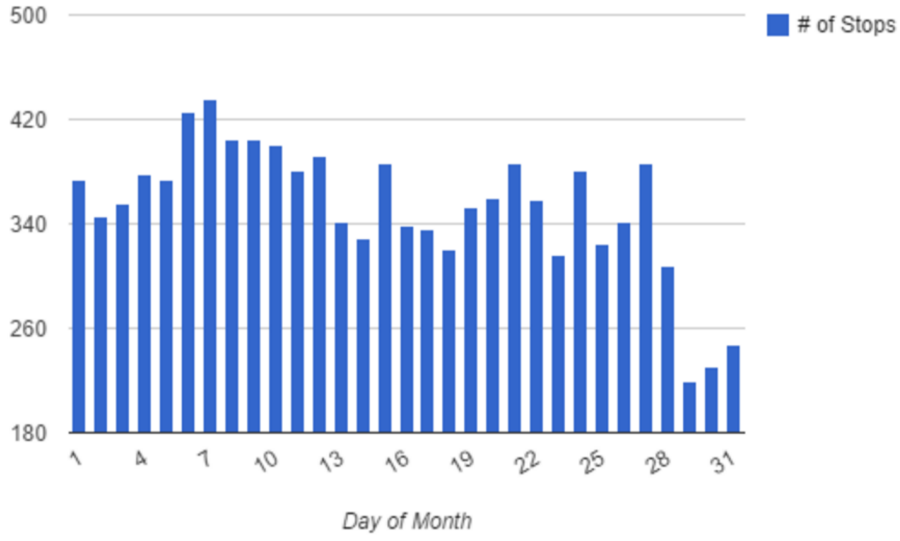


Figure 3: Distribution by Day of Month



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

Figure 4: Number of Stations per User

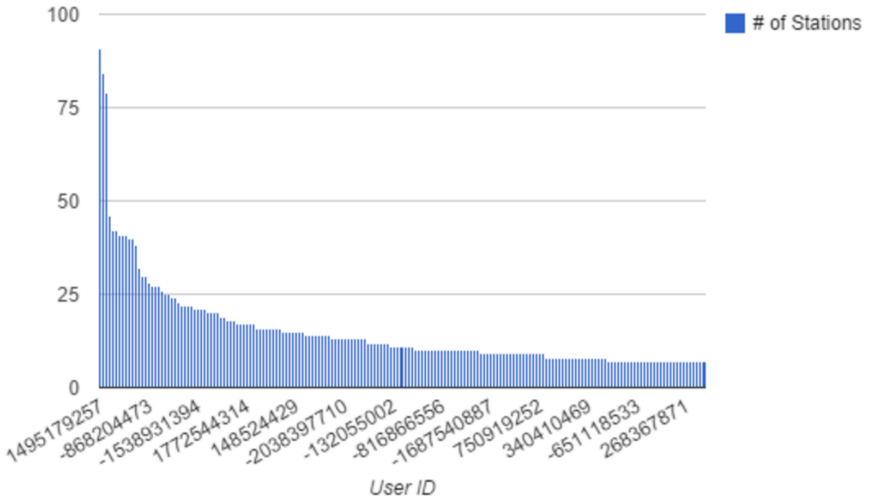
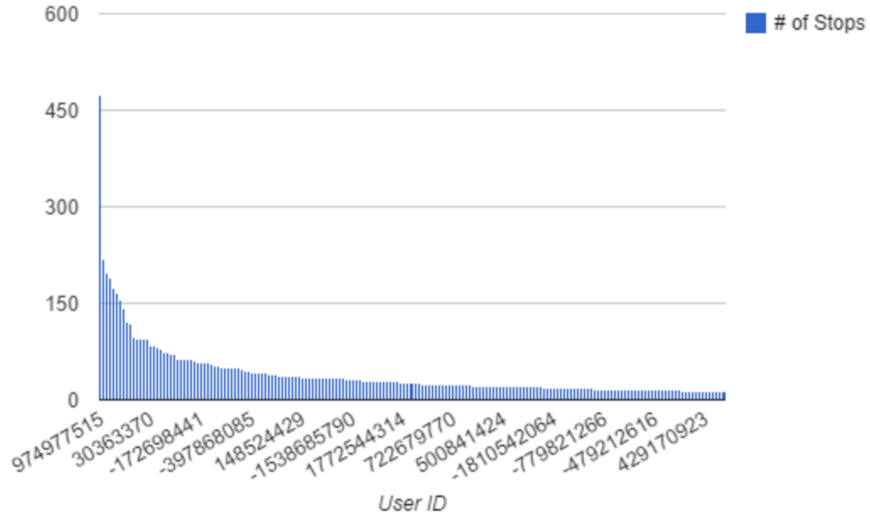
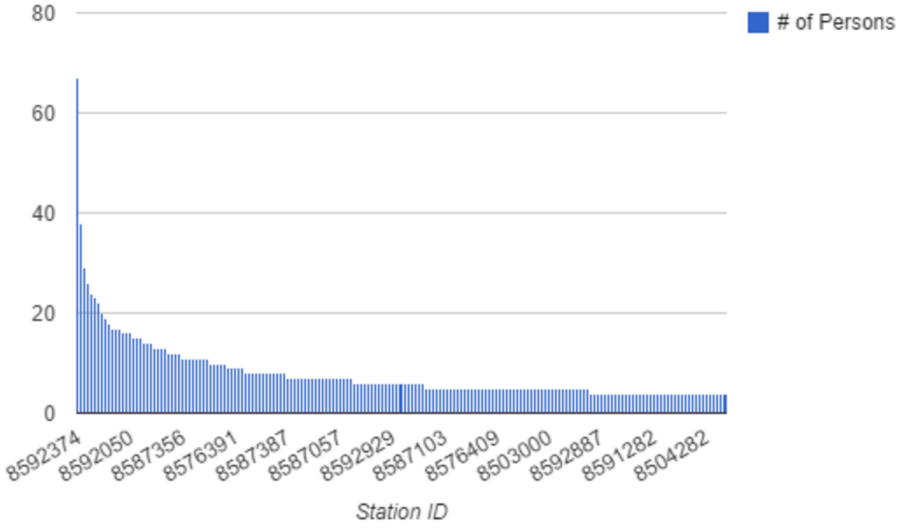


Figure 5: Number of Stops per User



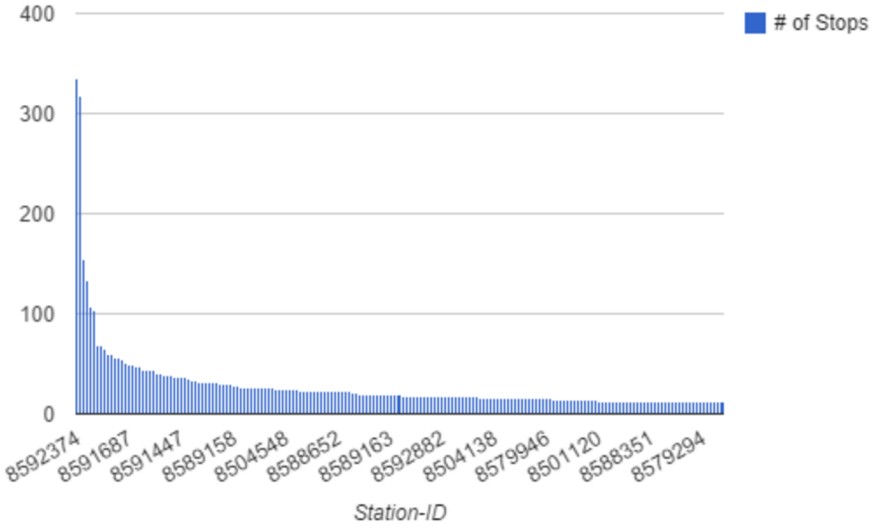
cases where a user really only has 2 or 3 stops that he regularly frequents this might work, otherwise it will just be stabbing in the dark to get a reasonably good prediction. One or two outliers from such a user could possibly mix up the complete prediction process. It seems sensible to cut the dataset 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,

Figure 6: Number of Users per Station



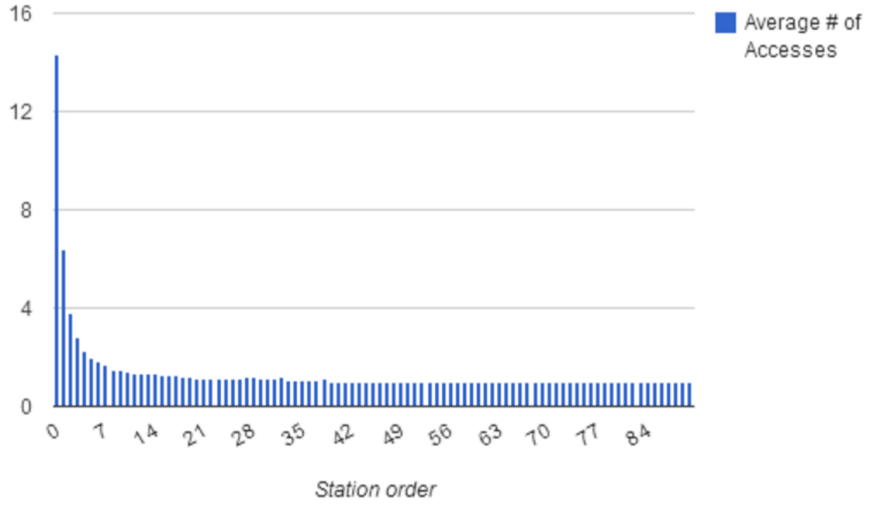
however if we would not cut the low-frequency user out of our comparison tests it might greatly change our conclusions and the effectiveness of our process.

Figure 7: Distribution of Stops per Station



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

Figure 8: Average number of accesses per station by users



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 5.4.2

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 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 created a graph that looked similar to —-Chart below—- 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.

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 50 percent (—- double check —-). 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.

5.3.1 Web GUI, Visual Representation

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.

5.4.1 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 4.2. We have used a Decision Tree (J48, unpruned —— WEKA Link ——) and a Naive Bayes classifier both provided from WEKA. As a classifier with back propagation we have used a Multilayer Perceptron classifier 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, be-

cause 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/weekendday 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 5.4.2.

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.

5.4.2 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:

```
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.

5.5 Machine Learning Results

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 [1]):

- 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.

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 & Timestamp**	**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

5.6.1 Comparison of all Users

Decision Trees

Naive Bayes

Multilayer Perceptron

5.6.2 Comparison of frequent versus non-frequent Users

Decision Trees

Naive Bayes

Figure 9: Stations-Feature for Decision Tree

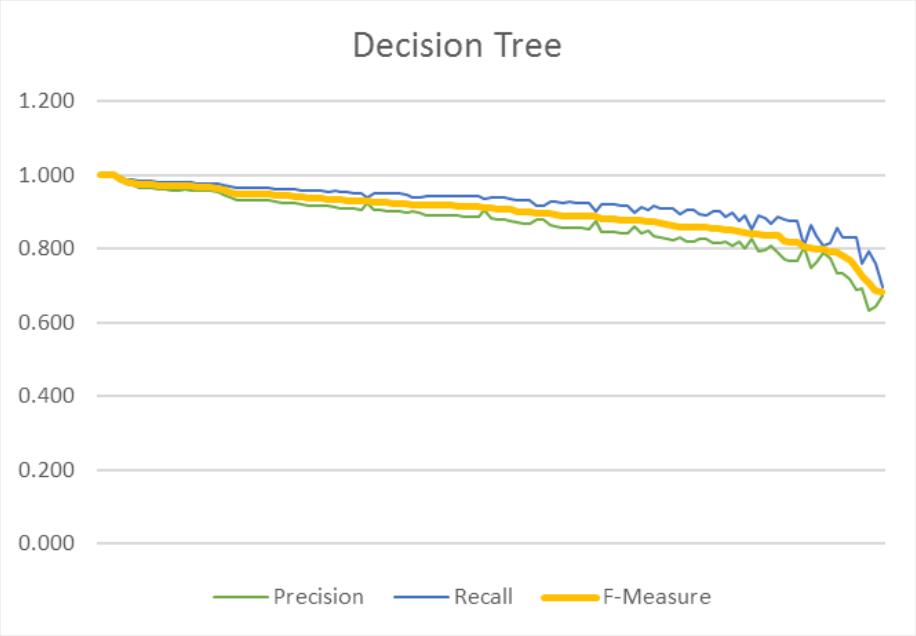


Figure 10: Stations-Feature for Decision Tree

Multilayer Perceptron

5.6.3 Comparison of all Users with higher number of Stations

Decision Trees

Naive Bayes

5.7 Comparison of Results

6 Conclusion

Here goes the conclusion.

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 could be included in the actual Android application where the data is gathered. If carefully and helpfully integrated into the application so as that the user actually benefits of it but doesn‘t consider it as too intruding in his privacy 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 also 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 and data preparation and also to the options of the machine learning algorithms might be taken. After our analysis phase and some trials on the data we have imposed relatively 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 to 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 might also 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 mostly based on neural networks and can be scaled from anything 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.

References

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

[2] Wikipedia. Decision tree learning, 2016.