A Route Recommendation System Based on Temporal Characteristics
Acknowledgments

Many people have contributed significantly in order to make this thesis possible. I am very grateful to every person’s assistance, no matter how big or small. In particular, I would like to thank:

My academic supervisor, Associate Professor Vasilios Vassalos, for excellent guidance and continuous support. All of our discussions were constructive and advanced the quality of this paper.

The company Beat, the employees Dimosthenis Kaponis and Ilias Antoniou. They were by my side the whole time, patiently listening and guiding me through every effort and hurdle. Ilias in particular had to go through my endless theory questions and practical suggestions to solve many issues. They’re always positive attitude made this cooperation a pleasant endeavor.

Family friend Christi Grab, who as a talented writer, editor and native English speaker, contributed in making the text so much clearer and easier to read.

My partner in life Matina and my parents Pantelis and Androniki for reading my text, making suggestions and bearing with me through the whole process.

Of course, every mistake present in this thesis is the sole responsibility of the author.

Ioannis Lakkas
Table of Contents

Acknowledgments........................................................................................................3
1. Introduction..................................................................................................................7
  1.1 The Problem.............................................................................................................7
  1.2 The Data....................................................................................................................9
  1.3 The Market...............................................................................................................10
  1.4 Approaching the Problem.......................................................................................11
2. Hidden Markov Model (HMM)..................................................................................13
  2.1 Training: Forward–Backward algorithm.........................................................17
  2.2 Training: Viterbi algorithm..........................................................18
  2.3 Future States Prediction.......................................................................................19
3. Utilizing HMMs in Destination Prediction...............................................................21
  3.1 Method 1: Modeling the Route Sequence.......................................................21
  3.2 Method 2: Modeling the Daily Repeatability.................................................25
  3.3 The Development of a Suggestion Metric.......................................................29
  3.4 The Steps of Data Processing................................................................................31
  3.5 Accuracy vs Training Chain Length...............................................................34
  3.6 Gaussian Mixture Models Effect on Suggestion Metric..................................40
Practical Guide to the Produced Code........................................................................41
Bibliography..................................................................................................................45
1. Introduction

This Master Thesis was realized during my MSc in Data Science in Athens University of Economics and Business and was made possible through cooperation with the software company Beat (https://thebeat.co/), previously known as Taxibeat.

Beat was founded in 2011 as a startup company in Athens, Greece. The aim of Beat was to provide an easy and fun way of calling for a taxi through a smartphone application. The user was able to pick a certain taxi driver for their ride, based on previous user reviews. When the ride was completed, the user could leave a personal review of the service.

1.1 The Problem

A user opens the smartphone application looking for a taxi ride. The Beat application, using the current location of the potential passenger, can suggest one or more termination points for the ride. Then the user either selects one of the recommended destinations, or ignores the recommendations and selects a new destination, then completes the order. Alternatively, the user can decide to abort the process, stop the order and exit the application.

The process above happens every time a user is about to make a new order for a taxi ride. How can such a system be built so it can learn and suggest destinations to the user?

Utilizing the user’s history of rides, their current location and time of day, it is being attempted to predict the destination.
Certain requirements must be met. The user should be rather systematic about a certain route\(^1\) they follow, preferably both in the time of the day and the days apart that a certain route occurs.

Destination prediction is not some sort of wizardry; it follows common sense rules. If a scholar studying a user’s history of rides can with rational thinking predict the destination, then it may be possible that the machine can do so, as well. In the case where a scholar could not attempt a prediction, deeper data mining, revealing behavioral and personal details of the user, would be needed for the recommendation engine. This is of course beyond this thesis.

A more concrete set of goals could be the ones below:

1. At least 85% prediction accuracy on suggested rides.
2. Suggest to the largest possible user base while maintaining Rule 1.
3. Maybe explore more/alternative models and algorithms to learn different kinds of patterns.

The system is responsible for updating tables that keep the necessary information needed by the smartphone application to display the suggestions.

---

\(^1\) See definition of a “user route” in page 10.
1.2 The Data

This Master Thesis was based on ride data of users of the Beat service in Lima, Peru. The rides described in the data start in the middle of January 2017 until the middle of July 2017 (six months of 2017).

The data received from Beat was in a CSV\(^2\) file format and was anonymized before being shared with the author. The user ID was obfuscated, as were the geographical coordinates of ride’s start and ending points. That made it impossible to identify any user in any way, such as finding someone’s home address, names or other sensitive data. The above information is considered private and is not distributed in any way.

The data was in table form. Each table row represents a single ride of a given user at a given time. The describing features of each ride are:

1. The User ID in hashed form
2. The Starting point Latitude and Longitude
3. The Destination Latitude and Longitude
4. The Timestamp (exact time and date) of the ride start

\(^2\) Comma Separated Values or CSV files with the ending “.csv”. A CSV file is a simple text file that stores one table of data.
**Definition of a “User Route”**

The definition of a “User Route” is a triplet containing two coordinates\(^3\): starting point pair, destination point pair and the user “ID.”

A user may have one hundred rides in their history but only fifteen “user routes.”

The name “user route” implies that rides with the same two coordinates occurring in two different users will count as two different “user routes.” One for User A and one for User B.

The data describe 13,260,820 records from 576,638 unique users and 7,891,656 unique “user routes.”

1.3 The Market

The reason why Lima data was chosen among all the markets Beat serves, is because Lima has an intricate interest as a market. It is *larger* than the Athens’ market and *less regulated*, and thus more flexible. Despite efforts to the contrary, Lima is still considered to be lacking adequate public transportation, especially in remote areas of the city and off working hours. This is a market where taxis flourish.

It should be noted that Lima’s governing body has not regulated taxi fares, so there is no standard fare. This void is being

---

\(^3\) Geographical coordinate i.e. the pair of latitude and longitude
filled by companies following individual pricing policies. The fare metering is passed to the application that in turn informs the user about the cost of the ride.

The application, during the filling of the new ride form from the user, has the ability to offer suggestions about the ride destination. An upgrade in the suggestion capabilities of the application can offer an advantage in markets such as Lima’s.

1.4 Approaching the Problem

Beat has already implemented a simpler destination recommendation system, which analyzes repeating patterns on a fixed weekly basis. Looking forward to advancing their recommendation system, Beat has been developing and testing a system that is planned to be put into production soon. The new system uses the Hidden Markov Model, which has a respectable suggestion metric accuracy of 85%. This Thesis aspires to be a continuation of Beat’s effort to design a destination recommendation system that will be both ideal for the user and useful for Beat, as well.

This thesis recommendation system has been developed on the basis of the *Hidden Markov Model*. Beat *hasn’t contributed code* to this Thesis, and the produced source code was based only on public and open source libraries of Python such as *Numpy, Pandas, Scikit-Learn, Scipy* and *HMMLearn*.

Using HMM in this implementation instead of simpler tools to check weekly patterns increases the capacity of prediction of patterns, whose periods of repetition can vary from every day to only few times a month.
2. Hidden Markov Model (HMM)

Hidden Markov Model or HMM is a member of the family of Markov Models. In probability theory and statistics, Markov Models are stochastic models that describe systems with the following properties:

1. A system can be described by its state and the transition from one state to another is stochastic.

2. The jump probabilities from the current state to a new one is only dependent on the current state. The past states do not affect the jump in any way.

In this Thesis we will deal with systems that have a finite number of discrete states.

Let’s consider a set of four states that describe the state of the weather in a certain location:

{Sunny, Cloudy, Rainy, Snowy}

Six consecutive days of summer could be:

Illustration 2.1, Markov Chain with the six day weather states

A record of some consecutive states of a Markov system is named “Markov Chain.” Above we have a Markov Chain of our system’s weather states.

It is rather obvious from the chain above that the state “sunny” is most likely to transition to an also “sunny” state. The proba-
bilities that describe those kind of transitions are called “transition probabilities” and are represented in a matrix N x N where N stands for the number of the possible states (in the example is 4x4). Below there is an example:

<table>
<thead>
<tr>
<th>From\To</th>
<th>Sunny</th>
<th>Cloudy</th>
<th>Rainy</th>
<th>Snowy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunny</td>
<td>0.80</td>
<td>0.12</td>
<td>0.07</td>
<td>0.01</td>
</tr>
<tr>
<td>Cloudy</td>
<td>0.45</td>
<td>0.30</td>
<td>0.24</td>
<td>0.01</td>
</tr>
<tr>
<td>Rainy</td>
<td>0.55</td>
<td>0.30</td>
<td>0.14</td>
<td>0.01</td>
</tr>
<tr>
<td>Snowy</td>
<td>0.60</td>
<td>0.15</td>
<td>0.24</td>
<td>0.01</td>
</tr>
</tbody>
</table>

*Table 2.1, Transition probabilities matrix*

In *Table 2.1* we see that when the weather is “sunny,” it is most likely to be “sunny” the next day, too. When it is “rainy,” it is most likely that the next day will be “cloudy.”

In cases where we cannot observe the system states directly, but we can observe states that *scholastically* relate to the states of the system, we model the problem using the *Hidden Markov Model*. This model is the tool we use to inference those hidden system states. From now on the states that we can observe will be named as the *visible states*, and the hidden to us system states will be named as the *hidden states*.

Let’s assume there is a scientist that is working in an underground but naturally ventilated facility. Due to their work they are unable to directly observe the sky and the weather conditions. However they are able to feel the temperature of the surrounding air. Let’s say the states visible to the scientist are the following:
The states of the weather themselves can’t be observed, ie. these are the hidden states now:

Visible States: {Hot, Chilly, Cold}

Hidden States: {Sunny, Cloudy, Rainy, Snowy}

By observing the visible states, they can infer the hidden states of the weather conditions. For example the state “hot” will most likely be observed when the weather is “sunny.” A six day observations can be:

Illustration 2.2, HMM Chain with unknown hidden states of the weather. The hidden states cannot be observed directly.

Assessing the observed states assumes that the most probable weather states were:

Illustration 2.3, HMM Chain with the assumed weather states included in the illustration. This depicts an accurately learned occurrence of hidden states.

In the above illustration, the hidden states were evaluated 100% successfully.
Given a state of the system that is hidden to the observer, we can define the probabilities of observing any visible state. Those probabilities are defined as “emission probabilities” or “output probabilities” and are represented in a matrix form. The matrix is of \( N \times M \) dimension where \( N \) is the number of the hidden states of the system and \( M \) the number of the visible states (in the example 4x3).

<table>
<thead>
<tr>
<th>Hidden\Visible</th>
<th>Hot</th>
<th>Chilly</th>
<th>Cold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunny</td>
<td>0.95</td>
<td>0.04</td>
<td>0.01</td>
</tr>
<tr>
<td>Cloudy</td>
<td>0.45</td>
<td>0.50</td>
<td>0.05</td>
</tr>
<tr>
<td>Rainy</td>
<td>0.20</td>
<td>0.70</td>
<td>0.10</td>
</tr>
<tr>
<td>Snowy</td>
<td>0.01</td>
<td>0.14</td>
<td>0.85</td>
</tr>
</tbody>
</table>

*Table 2.2, Emission probabilities matrix*

Usually —as is the case in our problem— we are aware of a chain of visible states, but we ignore the hidden states, the transition probabilities and the emission probabilities.

To train the model, we use as an input a chain of consecutive observed states (the visible states) and the model learns the following:

1. *Transition* and *Emission* matrices
   
   (2.1 Training: Forward–Backward algorithm)

2. The most probable chain of consecutive hidden states that created the observed states
   
   (2.2 Training: Viterbi algorithm)

Before the training begins, the user has to *define the size* of the hidden state set. Only then the learning algorithm can start
learning the above two points. The assignment of the hidden set size is based on our knowledge of the nature of the problem. Below we will further examine the process of training our HMM model.

2.1 Training: Forward–Backward algorithm

The first step to train our model is to infer the Transition and Emission matrices. Using the standard notation of literature, we name the Transition matrix as $A$ and the Emission matrix as $B$.

*From Daniel Ramage, Hidden Markov Models, Section Notes, page 8, definition of Forward–Backward algorithm is:*

We initialize $A$ and $B$ as random valid probabilities matrices where $A_{i0} = 0$ and $B_{0k} = 0$ for $i=1..N$ and $k=1..M$. $N$ stands for the number of hidden states and $M$ as the number of visible states. $T$ stands for the number of items/observations in the HMM chain where $t=1..T$.

Repeat until convergence:

**E-Step**: Run the Forward and Backward algorithms to compute $\alpha_i$ and $\beta_i$ for $i=1..N$. Then set:

$$y_t(i, j) := \alpha_t(i) A_{ij} B_{jx(t+1)} \beta_j(t+1)$$

**M-Step**: Re-estimate the maximum likelihood parameters as:

$$A_{ij} := \frac{\sum_{t=1}^{T} y_t(i, j)}{\sum_{j=1}^{N} \sum_{t=1}^{T} y_t(i, j)}$$
\[
B_{jk} := \frac{\sum_{i=1}^{N} \sum_{t=1}^{T} 1[x_t = u_k]y_t(i, j)}{\sum_{i=1}^{N} \sum_{t=1}^{T} y_t(i, j)}
\]

### 2.2 Training: Viterbi algorithm

The second step in training our HMM model is to infer the most probable sequence of hidden states given the Transition and Emission matrices and a sequence of visible states of a HMM chain.


```plaintext
function VITERBI (observations of len T, state-graph of len N) returns best-path
    create a path probability matrix viterbi[N+2,T]
    for each state s from 1 to N do ; initialization step
        viterbi[s,1] <- α₀,s * bᵢ(ο₁)
        backpointer[s,1] <- 0
    for each time step t from 2 to T do ; recursion step
        for each state s from 1 to N do
            viterbi[s,t] <- \( \max_{s'} \) viterbi[s',t-1] * αₛ⁺₁,ₛ * bᵢ(οₜ)
        backpointer[s,t] <- \( \arg\max_{s'} \) viterbi[s',t-1] * αₛ⁺₁,ₛ
    viterbi[qF,T] <- \( \max_{s} \) viterbi[s,T] * αₛ,q_f ; termination step
    backpointer[qF,T] <- \( \arg\max_{s} \) viterbi[s,T] * αₛ,q_f ; termination step
    return the backtrace path by following backpointers to states back in time from backpointer[qF,T]
```
We used:

\[ v_{t-1}(i) \] the **previous Viterbi path probability** from the previous time step

\[ \alpha_{ij} \] the **transition probability** from previous state \( q_i \) to current state \( q_j \)

\[ b_j(o_i) \] the **state observation likelihood** of the observation symbol \( o_i \) given the current state \( j \)

### 2.3 Future States Prediction

After training our model we are interested in attempting to predict the next few states of our system (*Illustration 2.4*). Below we will examine the steps we need to follow to predict one future state of the system.

*Illustration 2.4, Step 1: HMM Chain with the hidden and visible states we want to predict*

Using the Transition matrix, we can see that the most probable hidden state after “sunny” is also “sunny” (*Illustration 2.5*).

*Illustration 2.5, Step 2: HMM Chain with the hidden state predicted*
Using the Emission matrix we can see that the most probable visible state of a “sunny” hidden state is “hot” (Illustration 2.6).

Illustration 2.6, Step 3: HMM Chain with the visible state predicted

To predict more states we repeat the steps above, as many times, as the number of predictions needed.
3. Utilizing HMMs in Destination Prediction

3.1 Method 1: Modeling the Route Sequence

The first attempt in modeling the problem using HMMs was based on the idea that a user repeats a number of routes in a rather circular fashion. As such, a user’s rides could be:

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Date (D/M)</th>
<th>Time</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>Work</td>
<td>Monday 4/9</td>
<td>08:00</td>
<td>1</td>
</tr>
<tr>
<td>Work</td>
<td>Home</td>
<td>Monday 4/9</td>
<td>18:00</td>
<td>2</td>
</tr>
<tr>
<td>Home</td>
<td>Work</td>
<td>Tuesday 5/9</td>
<td>08:00</td>
<td>1</td>
</tr>
<tr>
<td>Work</td>
<td>Market</td>
<td>Tuesday 5/9</td>
<td>18:00</td>
<td>3</td>
</tr>
<tr>
<td>Home</td>
<td>Work</td>
<td>Wednesday 6/9</td>
<td>08:00</td>
<td>1</td>
</tr>
<tr>
<td>Work</td>
<td>Home</td>
<td>Wednesday 6/9</td>
<td>18:00</td>
<td>2</td>
</tr>
<tr>
<td>Home</td>
<td>Work</td>
<td>Thursday 7/9</td>
<td>08:00</td>
<td>1</td>
</tr>
<tr>
<td>Work</td>
<td>Home</td>
<td>Thursday 7/9</td>
<td>18:00</td>
<td>2</td>
</tr>
<tr>
<td>Home</td>
<td>Work</td>
<td>Friday 8/9</td>
<td>08:00</td>
<td>1</td>
</tr>
<tr>
<td>Work</td>
<td>Home</td>
<td>Friday 8/9</td>
<td>18:00</td>
<td>2</td>
</tr>
<tr>
<td>Home</td>
<td>Cinema</td>
<td>Saturday 9/9</td>
<td>20:20</td>
<td>4</td>
</tr>
<tr>
<td>Cinema</td>
<td>Home</td>
<td>Saturday 9/9</td>
<td>23:45</td>
<td>5</td>
</tr>
<tr>
<td>Home</td>
<td>Work</td>
<td>Monday 11/9</td>
<td>08:00</td>
<td>1</td>
</tr>
<tr>
<td>Work</td>
<td>Home</td>
<td>Monday 11/9</td>
<td>18:00</td>
<td>2</td>
</tr>
<tr>
<td>Home</td>
<td>Work</td>
<td>Tuesday 12/9</td>
<td>08:00</td>
<td>1</td>
</tr>
<tr>
<td>Work</td>
<td>Market</td>
<td>Tuesday 12/9</td>
<td>18:00</td>
<td>3</td>
</tr>
<tr>
<td>Home</td>
<td>Work</td>
<td>Wednesday 13/9</td>
<td>08:00</td>
<td>1</td>
</tr>
<tr>
<td>Work</td>
<td>Home</td>
<td>Wednesday 13/9</td>
<td>18:00</td>
<td>2</td>
</tr>
<tr>
<td>Home</td>
<td>Work</td>
<td>Thursday 14/9</td>
<td>08:00</td>
<td>1</td>
</tr>
<tr>
<td>Work</td>
<td>Home</td>
<td>Thursday 14/9</td>
<td>18:00</td>
<td>2</td>
</tr>
<tr>
<td>Home</td>
<td>Work</td>
<td>Friday 15/9</td>
<td>08:00</td>
<td>1</td>
</tr>
<tr>
<td>Work</td>
<td>Home</td>
<td>Friday 15/9</td>
<td>18:00</td>
<td>2</td>
</tr>
<tr>
<td>Home</td>
<td>Cinema</td>
<td>Saturday 16/9</td>
<td>20:20</td>
<td>4</td>
</tr>
<tr>
<td>Cinema</td>
<td>Home</td>
<td>Saturday 16/9</td>
<td>23:45</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3.1: Fictional user data to illustrate a user with perfectly cyclical routes. The labels “home,” “work,” “market” and “cinema” are used to make the table.
simpler to the reader. In the recommendation engine, geographical coordinates are used.

As it may be apparent to the reader, the data of Table 3.1 shows two identical weeks. Note: A week in this thesis will always be from Monday to Sunday. The data of Table 3.1 can be predicted accurately with HMMs, a model representation of the above data is:

```
1 2 1 3 1 2 1 2 1 2 4 5
```

*Illustration 3.1, A HMM chain of one week’s rides. For clarity the second week has not been included in the illustration. The route ID has been set as the visible state. In reality, when you are training the model, all available data are used from the last recorded date and back to a certain start date or a certain number of days back.*

In *Illustration 3.1* we used as visible states the ID of each route in the way the user executed them. The hidden states are unknown to us. The hidden states attempt to roughly model mathematically the decision making state of the user such as personal obligations, work, family, friends, acquaintances, hobbies and so on...

In the example above, with a sufficient amount of hidden states, the model can perfectly learn all the sequences of the rides. That way every next ride can be predicted.

The rides of the Table 3.1 make a complete circle every twelve rides. By setting the hidden states of our model to twelve we are certain that our model has sufficient complexity and can learn the given patterns.
Let’s now assume we’ve set the hidden states to just two, then the training on the data above would assign a hidden state to route ID 1 and a second hidden state to route ID 2. Training that model results in a prediction of a route ID chain like the following:

1 → 2 → 1 → 2 → 1 → 2 → 1 → 2 → 1 → 2 → 1 → 2 → 1 → 2...

instead of the correct one:

1 → 2 → 1 → 3 → 1 → 2 → 1 → 2 → 1 → 2 → 1 → 2 → 4 → 5 → 1 → 2 → 1 → 2 → 1 → 3...

The result above is based on the fact that the transition probabilities of the hidden states one and two would follow a switching pattern so it can correctly learn the majority of the visible states (it would only correctly learn nine of the twelve states of the chain). Of course, with such a few hidden states, routes three, four and five would not be learned, even if there were part of a perfectly repeating pattern.

Unfortunately, this method (Method 1) is flawed at the level of conception. It requires that users habitually execute routes in a circular fashion. This, of course, doesn’t match reality. While a user repeats usual rides such as routes to and from work, home or leisure, there will always be a few rare and unmatched routes. For example, a visit to a doctor or a distant relative. Such ride histories are aperiodical or have a period that is equal to the number of rides.

In those rather common cases the model attempts to learn the whole chain, not just patterns or pieces of it. By learning the whole chain, the result is that it fails to recognize any patterns. This is a textbook example of overfitting. One modification
could be to remove routes that show very few rides every month.

Another issue for consideration is the computational complexity. Training a model with thirty or fifty or more hidden states is substantially slower than a model with seven or fourteen hidden states. The large number of hidden states is mandated by the nature of the chains. Often the chains have lengths of one hundred or more elements. Training such chains becomes tedious and the model capacity of learning the data usually shows poor performance. As a consequence, predictive qualities are poor, too.

There’s also the element of time that this model totally ignores; certain rides happen during certain time periods. If one morning the user does not follow their typical work route, he likely won’t work at all that day, possibly due to illness or another obligation. This issue makes these sequences not particularly reliable and robust. Their predictive abilities are poor because of that.
3.2 Method 2: Modeling the Daily Repeatability

One way to avoid very long chains of route sequences and to incorporate the temporal element of the data sequence is to create a HMM chain for each user route. The visible state will be whether a route was executed on a certain day.

Analyzing the data of Table 3.1, we find that the user follows five unique routes, as shown in Table 3.2. Using the second method of modeling the data, we create five HMM models, one for each unique route.

<table>
<thead>
<tr>
<th>ID</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Home</td>
<td>Work</td>
</tr>
<tr>
<td>2</td>
<td>Work</td>
<td>Home</td>
</tr>
<tr>
<td>3</td>
<td>Work</td>
<td>Market</td>
</tr>
<tr>
<td>4</td>
<td>Home</td>
<td>Cinema</td>
</tr>
<tr>
<td>5</td>
<td>Cinema</td>
<td>Home</td>
</tr>
</tbody>
</table>

*Table 3.2, The user routes based on the Table 3.1.*

To better illustrate the concept of the second method, we will draw all of the above five routes into HMM chains below. The visible states get the values 0 or 1, 0 stands for “didn’t follow the route that day” while 1 for “followed the route that day.”

The illustrations below show only one week of data for clarity. When coding chains like these, all training data for each route would be included in the chains.
Illustration 3.2, Preview of the modeling of the route 1 “home – work.” It is obvious from both Table 3.1 and the illustration that the route is being followed only on working days.

Illustration 3.3, Preview of the modeling of the route 2 “work – home.” It is obvious from both Table 3.1 and the illustration that the route is being followed only on working days except Tuesdays. On Tuesdays, the user goes to the Market after Work and then Home.

Illustration 3.4, Preview of the modeling of the route 3 “work – market.” It is obvious from both Table 3.1 and the illustration that the route is being followed only on Tuesdays.
Illustration 3.5, Preview of the modeling of the route 4 “home – cinema” and 5 “cinema – home.” It is obvious from both Table 3.1 and the illustration, the route is being followed only on Saturdays.

Predicting the next item’s \textit{(the item next to the last of the chain)} hidden and visible states, can figure out if the next day’s ride will be executed or not. This way, we learn all near future dates a ride is likely to happen. Usually, a sufficient number of hidden states for this model is seven. Most routes with rather frequent rides show weekly patterns, or periods of seven days, justifying the seven hidden states. Routes with less frequent rides (routes that repeat every few weeks) certainly need more hidden states to be learned.

The repeatability period of the data can be effectively inferred using autocorrelation. Through this process we learn the most dominant periodicity of the data. The most dominant periodicity of the data is then assigned as the number of the hidden states the HMM model has to learn.

The next step of the process is to learn the \textit{starting times} of the rides in every route. In order to learn the starting times, we have to assess the route’s history of starting times. We expect the predicted ride’s start time to fall into the same \textit{time frame} of the historical starting times. Finally, if a user opens the application in one of the known \textit{starting points} in the same time
frame they usually take a ride in, then this route will be suggested.

One crude way to define a time frame is to average the route’s starting times and assess if the time in question belongs in the historical data within one or more standard deviations.

This crude way works rather well in unimodal distributions of route starting times.

Illustration 3.6, Typical unimodal route starting times, starting ride hour vs rides starting in that hour

However, there are real user starting times that are multimodal, a user may follow a route in the midday and in the evening but never in the afternoon.

Illustration 3.7, Typical multimodal route starting times, starting ride hour vs rides starting in that hour. Using GMM each peak is modeled after a normal distribution, displayed in different color

Such route times can be learned with the Gaussian Mixture Models. In each time peak a normal distribution is assigned. The parameters of the normal distributions are learned by the
Gaussian Mixture Models algorithm. How GMM time analysis affects the Suggestion Metric will follow in chapter 3.6 Gaussian Mixture Models Effect on Suggestion Metric, in page 40.

### 3.3 The Development of a Suggestion Metric

Building on the paradigm of the second method of modeling our data, Beat suggested a metric that incorporates the logic of the problem we are trying to solve and judges the prediction quality based on the behavior of real users.

Let’s consider a user that is currently in *Place A*. Let’s examine a certain user route that starts from *Place A* and goes to *Place B*. Let’s consider the three possible events:

- **Event 0**: The user does not get a taxi from Place A (they close the app without any further actions)
- **Event 1**: The user *does* get a taxi ride from Place A to Place B
- **Event 2**: The user gets a ride from Place A to a place other than B — let’s call it “*Place C*”

So there are three possible events for a user, then there is the *Actual* choice a user makes and the *Predicted* choice that the prediction engine gave beforehand. Placing all the above in a table and we get the following:
The Table 3.3 can be summarized as:

- No prediction made
  Outcome: Neutral

- There was a prediction but the user did nothing
  Outcome: Neutral
  (maybe the user had an obligation and didn’t follow the route or used another means of transportation)

- The user followed the route suggested
  Outcome: Success

- The user followed another route than the one suggested
  Outcome: Fail
3.4 The Steps of Data Processing

Below I will try to roughly describe the steps of the data processing pipeline from end to end, from the initial data loading to the creation of the route recommendations.

1. **Stage 1: CSV pre-process:**
   
   Read the input CSV with the following headers:
   
   id_passenger, src_lat, src_lng, dest_lat, dest_lng, timestamp
   
   Output a CSV with the following headers:
   
   id_passenger, route, source, dest, timestamp, month, day, hour, weekday

2. **Stage 2: Filter Stage 1 CSV routes and keep only those with three or more rides, then save the data as Stage 2 CSV.** Optionally, instead of saving the data into a single large CSV output, break it into multiple Stage 2 CSVs. The number of files being created is defined by the user and the resulting CSVs have similar sizes.

3. **Stage 3: Learn each Stage 2 CSV in a different CPU process.**

   For example, if a server has eight CPU cores available, we create eight Stage 2 CSVs and start eight python instances. Each instance learns from one of the eight CSVs. Each python instance produces a Route Recommendation CSV and a Suggestion Metric CSV.

The Route Recommendation CSV has the following headers:

id_route, service, seasonality, seasonality_range, id_passenger, from_address, from_lat, from_lng, to_address, to_lat, to_lng, from_tod, to_tod, frequency, score, deleted_at, created_at, weeks_back

The Suggestion Metric CSV has the following headers:

id_passenger, route, exact_prediction, tb_metric, train_days, train_rides, train_density, ipos
4. **Stage 4**: Read all *Route Recommendation CSVs* and *Suggestion Metric CSVs*, and merge them into two large CSV files.

5. **Stage 5**: Pass the final Route Recommendation CSV to the *Beat* server infrastructure so the suggestions appear in the smartphone application.

*(Not planned to be part of this Thesis)*

In this Thesis, the data of each route was split to **80% training data** and **20% test data**. If, for example, a route had trips spanning seventy days or ten weeks, the first 56 days or eight weeks would be training data and the following fourteen days or two weeks would be test data. Routes with rides spanning between two to nine weeks have the last week as test data, routes with data spanning ten or more weeks have the last two or more weeks as test data.

It should be noted that the **Stage 3** (learning user routes from **Stage 2 CSVs**) process can be interrupted at any time without loss of previous calculations. The user of this thesis code can restart the process to start learning again at a later time from any given point.

The **Stage 4** process can merge route recommendation parts from both:

- Split output files due to parallel instances running
- Split output files due to user interrupts and continuations

For example, a user of this thesis code has split the data to eight parts so eight threads can work concurrently. When the
processes reach about 40%, the user of this code decides to interrupt the process to devote the processing power elsewhere. Afterwards, they continue the learning process from the spot that was interrupted until completion. The result is sixteen Route Recommendation CSVs and a Suggestion Metric CSV that have to be merged into two CSV files. This is handled automatically by the Stage 4 process. This process removes the duplicates and keeps only a single version, the version with the best Suggestion Metric.
3.5 Accuracy vs Training Chain Length

Designing a machine learning system usually requires some basic optimization that takes into consideration the nature of the data. In our case, there are users that follow certain routes for many months; for these routes we may have data that spans over twenty weeks. We can certainly use all available data for such routes and train our model with them. Training a route with twenty weeks of data is certainly slower than training a model with ten weeks of data or less. Can the length of the training chain affect the accuracy of the suggestions?

Consistent users have data that follow repeating patterns. Learning one instance of such patterns is enough to accurately learn the user’s habits for that route. In reality, most users are rather consistent over short periods of time, but as seasons change, they modify their transportational habits. Holidays affect and change the users habits, too.

The ideal training data should contain the minimum amount of data that contain the complete behavioral footprint of the user. Recent data does hold a more accurate description of a user’s behavior.

For example let’s consider a user route that has data spanning twenty weeks. We decide to use the last two weeks as the test set and for the training set make the two following cases:
1. Use *All* Data available:

Illustration 3.8, Twenty weeks of data. The eighteen weeks of training data are in green, and the last two weeks of test data are in red. The numbers show the time in weeks before present time.

2. Use only *Recent* Data:

Illustration 3.9, Twenty weeks of data. The eight weeks of training data are in green and the last two weeks of test data are in red. White cells show data that was discarded and not used in the learning process. The numbers show the time in weeks before present time.

In the above two cases, the second case usually gives a better Suggestion Metric.

Let’s now consider the *Average Suggestion Metric* of routes spanning *up to* a certain number of weeks:

Illustration 3.10, The Average Suggestion Metric of routes that their training data span *up to* a certain number of weeks. As we move from the left side of axis X to the right, more and more routes are getting included in the calculation of the Average Suggestion Metric. In the far right of the graph all available data of all routes are being included in the calculation (training time of more than five months). In the right, you see that the Average Suggestion Metric is just shy of 85%.
It is obvious from *Illustration 3.10* that allowing longer training chains lowers the *Average Suggestion Metric* of the predictions.

Let’s now consider the *Average Suggestion Metric* of routes spanning exactly a certain number of weeks:

![Illustration 3.11, The Average Suggestion Metric of routes with training data spanning exactly a certain number of weeks.](image)

*Illustration 3.11* shows that training chains with lengths from eight up to twenty weeks show a considerable dip in the accuracy.

To further explain *Illustration 3.11* above, we will try to break the data into multiple plots. We will again plot the *Average Suggestion Metric* vs the *Training Weeks* as above, but filtered for routes that ended in a certain time window of two weeks.
Training ended 1 or 2 weeks before July 16
Training ended 3 or 4 weeks before July 16
Training ended 5 or 6 weeks before July 16
Training ended 7 or 8 weeks before July 16
Training ended 9 or 10 weeks before July 16
Training ended 11 or 12 weeks before July 16
Training ended 13 or 14 weeks before July 16
Training ended 15 or 16 weeks before July 16
Training ended 17 or 18 weeks before July 16
Training ended 19 or 20 weeks before July 16
Illustration 3.12, Same visualization as Illustration 3.8 but only showing routes that the training date ended by the number of weeks, mentioned in the caption.

The dates of Illustration 3.12 can be found in the table below:

<table>
<thead>
<tr>
<th>Illustration 3.10 plot →</th>
<th>Week ending at</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-2</td>
</tr>
<tr>
<td>2017-07-16</td>
<td></td>
</tr>
<tr>
<td>2017-07-09</td>
<td>1</td>
</tr>
<tr>
<td>2017-07-02</td>
<td>2</td>
</tr>
<tr>
<td>2017-06-25</td>
<td></td>
</tr>
<tr>
<td>2017-06-18</td>
<td>4</td>
</tr>
<tr>
<td>2017-06-11</td>
<td></td>
</tr>
<tr>
<td>2017-06-04</td>
<td>6</td>
</tr>
<tr>
<td>2017-05-28</td>
<td>7</td>
</tr>
<tr>
<td>2017-05-21</td>
<td>8</td>
</tr>
<tr>
<td>2017-05-14</td>
<td>9</td>
</tr>
<tr>
<td>2017-05-07</td>
<td>10</td>
</tr>
<tr>
<td>2017-04-30</td>
<td>11</td>
</tr>
<tr>
<td>2017-04-23</td>
<td>12</td>
</tr>
<tr>
<td>2017-04-16</td>
<td>11</td>
</tr>
<tr>
<td>2017-04-09</td>
<td>12</td>
</tr>
<tr>
<td>2017-04-02</td>
<td>13</td>
</tr>
<tr>
<td>2017-03-26</td>
<td>14</td>
</tr>
<tr>
<td>2017-03-19</td>
<td>15</td>
</tr>
<tr>
<td>2017-03-12</td>
<td>16</td>
</tr>
<tr>
<td>2017-03-05</td>
<td>17</td>
</tr>
<tr>
<td>2017-02-26</td>
<td>18</td>
</tr>
<tr>
<td>2017-02-19</td>
<td>19</td>
</tr>
<tr>
<td>2017-02-12</td>
<td>20</td>
</tr>
<tr>
<td>2017-02-05</td>
<td>19</td>
</tr>
<tr>
<td>2017-01-29</td>
<td>20</td>
</tr>
<tr>
<td>2017-01-22</td>
<td>21</td>
</tr>
<tr>
<td>2017-01-15</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 3.4, The dates of Illustration 3.12. The red dates mark the Easter period, Sunday 2017-04-16 was Easter Sunday and 2017-04-09 was Palm Sunday.

By looking at both the Illustration 3.12 and the Table 3.4, we can see that having short periods of training always results in the best possible Suggestion Metric. Longer training periods
can have a smaller or larger detrimental impact on the Suggestion Metric depending on the time period included. As it is visible, training weeks that contain the Easter season (shown in the table in red) have a strong negative impact in the Suggestion Metric. This is more pronounced in the longer training sets. Even “15-16 weeks” and “17-18 weeks” plots are being adversely affected due to the test set lying during the holidays. Plots unaffected by Easter (in January, February, May, June and July) show good a Suggestion Metric. Up to seven weeks of training seems to be the sweet-spot in acquiring a good Suggestion Metric result.

Below are the performance results of the proposed suggestion engine:

<table>
<thead>
<tr>
<th>Run</th>
<th>Avg Suggestion Metric</th>
<th>Route Recommendations</th>
<th>Routes with Four or More Rides</th>
<th>Percent of Routes Suggested</th>
<th>Total Routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Six Months, All weeks available in Training</td>
<td>84.6%</td>
<td>181,537</td>
<td>450,270</td>
<td>40%</td>
<td>7,891,656</td>
</tr>
<tr>
<td>Six Months, Up to 7 weeks in Training</td>
<td>90.4%</td>
<td>157,691</td>
<td>450,270</td>
<td>35%</td>
<td>7,891,656</td>
</tr>
<tr>
<td>Last 70 days of data, All weeks available in Training</td>
<td>91.4%</td>
<td>117,658</td>
<td>201,734</td>
<td>58%</td>
<td>3,848,178</td>
</tr>
</tbody>
</table>

*Table 3.5, Performance results for the different sets of training lengths.*
3.6 Gaussian Mixture Models Effect on Suggestion Metric

Building on the introduction of Gaussian Mixture Models usage in learning the user’s typical route times (chapter 3.2 Method 2: Modeling the Daily Repeatability, page 25), we present the performance results.

<table>
<thead>
<tr>
<th>Run</th>
<th>Avg Suggestion Metric</th>
<th>Route Recommendations</th>
<th>Routes with Four or More Rides</th>
<th>Percent of Routes Suggested</th>
<th>Total Routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Six Months, Up to 7 weeks of Training, GMM time analysis 95% confidence interval in each normal of the model</td>
<td>90.4%</td>
<td>157,691</td>
<td>450,270</td>
<td>35%</td>
<td>7,891,656</td>
</tr>
<tr>
<td>Six Months, Up to 7 weeks of Training, assuming times follow normal deviation 95% confidence interval</td>
<td>89.8%</td>
<td>131,661</td>
<td>450,270</td>
<td>29%</td>
<td>7,891,656</td>
</tr>
</tbody>
</table>

*Table 3.6, Performance results with and without Gaussian Mixture Models starting times analysis.*

It is obvious that the advantage offered by the GMM starting time analysis is subtle but noticeable. The Average Suggestion Metric has less than a one percent advantage, but the routes suggested increased by about twenty percent over the simple model.
Practical Guide to the Produced Code

The core files of the code can be narrowed down to the following:

1. `dataset_preprocess_stage1.py`
2. `dataset_preprocess_stage2.py`
3. `libtaxibeat.py`
4. `automated_hmm_taxibeat.py`
5. `qt5_hmm_taxibeat.py`
6. `libvisualize.py`
7. `results_exploration_stage4.py`
8. `results_exploration_stage4_2.py`

Files one and two implement the data pre-processing Stages one and two of the pipeline (see page 31).

File three, “`libtaxibeat.py`”, incorporates core routines for calculations such as:

- Autocorrelation
- Gaussian Mixture Model analysis of route start times
- Selection of test and training dates
- Creation of Training and Test dataframes
- Implementation of the Suggestion Metric
- HMM model check routines
File four, “automated_hmm_taxibeat.py” implements a batch version of the destination recommendation system, uses “libtaxibeat.py” for the calculations. Outputs one CSV file of destination suggestions and one CSV file containing the learning performance report (more in page 31).

File five, “qt5_hmm_taxibeat.py” implements a GUI version of the destination recommendation system, uses “libtaxibeat.py” for the calculations and “libvisualize.py” for the data plotting. Based on Qt5 UI toolkit, the GUI version was developed before the batch version and was employed in an effort to quickly and effectively develop the learning engine while identifying and ironing out any bugs quickly.

File six, “libvisualize.py”, implements routines that preview matplotlib plot objects into the Qt5 interface. Using the above library, two visualizations are being created:

- Prediction vs Test data plot
• Start time barplot.

File seven, “results_exploration_stage4.py”, post-processes the output CSVs from multiple runs of “automated_hmm_taxibeat.py”, combining them into just two CSV files. One CSV file of destination suggestions and one CSV file containing the learning performance report (more in page 31)

File eight, “results_exploration_stage4_2.py”, reads route data and adds columns of statistical data into the performance report made by “results_exploration_stage4.py”.
Bibliography

Books:


Papers:


Presentations:

1. Zoubin Ghahramani, Jurgen van Gael, Yee Whye Teh, Yunus Saatci, *Nonparametric Bayesian times series models: infinite HMMs and beyond*, University of Cambridge

2. Zoubin Ghahramani, Matt Beal, Jurgen van Gael, Yunus Saatci, Tom Stepleton, Yee Whye Teh, *Bayesian Hidden Markov Models and Extensions*, Department of Engineering, University of Cambridge