# Personalized Navigation: Context-Based Preference Inference Using TensorFlow

Joachim Klokkervoll, Samuel Nygaard, and Mike Pedersen
{jklokk12, snpe12, mipede12}@student.aau.dk

June 6, 2017

## Abstract

Existing navigation systems provide route suggestions with very limited room for customization or personalization, and if present, must be configured manually by the user. We propose a context-based personalized navigation framework capable of performing personal route suggestion based on a preference towards certain road features, with the preference inferred from previously driven trips.

The framework infers a small set of unique driving preferences, and use contextual information to learn personal preference choice in the given context. This allows for generalization of preferences, fast learning, and good scalability while allowing for context-based personalization within the globally optimal preference set.

Training data stems from the ITS project and after filtering consists of 958 thousand driven routes mainly in North Jutland, Denmark from 458 drivers. We also use a speedmap of North Jutland provided by Aalborg University to infer speed information about roads. Lastly, OpenStreetMap is used to extract appropriate road features and construct a graph structure used for pathfinding.

After training with a large real-world dataset, our framework can propose personal route suggestions for future navigation trips with an average Jaccard distance of 0.48 when compared to the actual driven route preferred by the driver. The framework is trained using a combination of clustering and classification implemented in TensorFlow. The preferences are learned by comparing driven routes to alternative routes.

## 1 Introduction

Today's leading navigation services do not take the users' preferences into account when performing route suggestions, but instead provide impersonal routes between two locations with the routes affected only by simple contexts (e.g. traffic and time of day). Drivers could benefit from a more tailored route that matches their preferences.

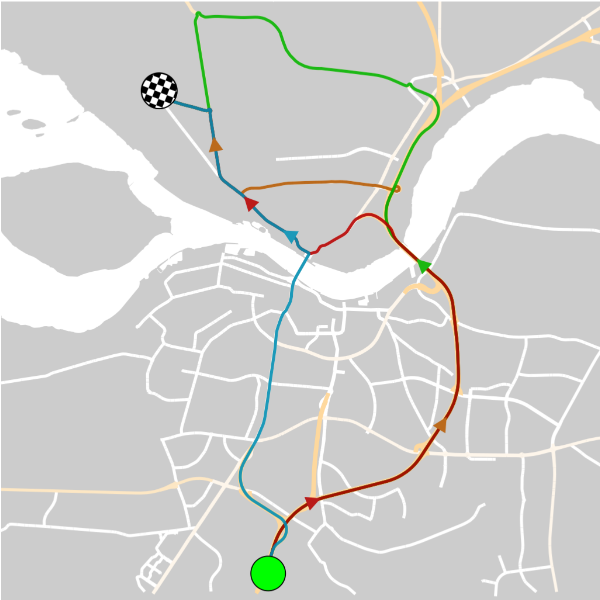


Figure 1: Four real-world routes having the same source and target. The one preferred by a driver might depend on several contexts (e.g. time of day and weather).

As an example, a driver that tends to avoid tunnels, either in general or in specific contexts, would benefit from a navigation service based on personal driving preferences, as it could learn such preferences, adjust pathfinding costs accordingly, and suggest alternative routes with fewer tunnels. In this paper, the concept "shortest path" has a more general meaning than just shortest distance, namely the cheapest/lowest cost of a path.

Comprehensive research has been published in the field of personalized navigation systems [2, 3, 4, 5, 6]. Most of this research is concerned with extracting personal driving preferences from GPS records using various machine learning techniques. However, these studies mainly examine the driving preferences without any contextual information such as weather or time information.

In this paper, we present a framework for providing contextual personalized routing based on the concept of driver preference towards certain road features, with the preference learned from the driver's previous trips.

In addition, we introduce the concept of a *context*. The context of a trip is the time of day, day of week, weather, etc., that may influence how a driver decides on a route. This can either be due to the road features changing (e.g. rush hour), or due to the preferences changing at different times (e.g. a preference for larger roads when it is snowing).

In [1], a preference is defined as a vector with a weight for each road feature and determines the cost of traversing a road segment. These road features are static, i.e. the features of the road segment never change. In this paper, we extend this definition to include dynamic features which change based on the context, e.g. estimated road segment speed which changes over time.

Furthermore, we extend the personal navigation framework presented in [1] to include clustering. Assembling trips with similar preferences into clusters allows for faster shortest path computation and instead of encapsulating a unique driving preference for each driver we try to obtain driving preferences for a group of drivers.

As seen in Figure 1 a single driver might choose different routes between the same two locations depending on the current context of their situation. In the example, the routes call for different preferences, with some (red, green, brown) preferring motorways and longer routes to different degrees and others (blue) having a higher preference towards shorter, but slower, routes.

The contributions of this work are as follows:

- We present the idea of personal preference vectors from Klokkervoll et al. [1] and extend it with contextual classification.
- We utilize clustering to improve the run time and scalability of the framework and to better accommodate new users with few driven routes by bootstrapping them with the preferences of similar drivers.
- We present several experiments for evaluating the usefulness of such a framework.

The paper is structured as follows: Section 2 outlines the related work in this problem area, Section 3 formalizes the modeling of the problem presented in this paper, and Section 4 explains the concept of personalization. In Section 5 we describe our dataset and present the different experiments we conduct.

# 2 Related Work

To the best of our knowledge only one other article, [7] by Yang et al., has previously examined context-based personalized navigation. However, the field of personalized navigation have been comprehensively examined by several researchers.

In this section, we will give an overview of the existing work in each category: Personalized navigation and context-based navigation.

## 2.1 Personalized Navigation

Delling et al. [2] delve into the realm of personalized navigation while utilizing GPUs for machine learning. The machine learning techniques used are: Local search (a basic stochastic coordinate descent), perturbation and specialized sampling. Delling et al. use a rather simple quality score where they essentially compare the shortest Euclidean length between each observed GPS record with a computed shortest path (from the source to the target). If, for instance, half of the GPS records is within the distance threshold the computed shortest path would get a quality score of 0.5, they utilize their framework using a threshold distance of 10 meters. We differ from this paper by utilizing another personalization method as well as clustering.

Funke et al. [3] presents a framework for deducing individual driving preferences utilizing a linear programming (LP) formulation combined with an effective shortest path algorithm, namely customizable contraction hierarchies, which we also utilize in this paper. Funke et al. defines the term *preferential feasibility* to describe if a set of paths can be exactly described with a preference vector (weight vector) $\alpha$. The LP-formulation is defined by the road features of any alternative path and therefore requires one to enumerate over all alternative paths from the source to the target, for which there are typically exponentially many. To overcome this challenge, Funke et al. improves the LP-formulation by starting with a few constraints, finding a solution, and the iterating by using that solution to provide new constraints. This LP-formulation uses hard constraints, i.e. it will find a preference that exactly matches the route. Instead, we use soft constraints and a gradient descent optimization method. These soft constraints allow routes that approximates the true route in the case where the true route is preferentially infeasible. The dataset used by Funke et al. is exclusively synthetically generated. In an experiment, Funke et al. achieves 0-2% difference in road segments between the ground truth routes and the routes of the inferred preference vector. Using CCH for shortest path calculation improved the run time with 2 orders of magnitude.

Letchner et al. [4] present a system, TRIP, able to do personalized routing based on data from the Seattle municipally. They present the first ever framework based on real data gathered from 109 cars using 8 GPS devices installed for a two-week period in each car, which is similar to the format of our data. A novel contribution at the time was the time-dependent speeds in the road network inferred from the observed GPS records. The presented solution uses a cost function based on the inefficiency ratio, which is the ratio between the route at a given time and the actual driven route for each route. This inefficiency ratio is applied, per driver, to all road segments that the driver has used and can be described as a measure of how much additional travel time a driver is willing to use to travel on these road segments, or how preferred they are to the driver. Letchner et al. define a cost measure using this ratio so that the cost for roads with a low ratio is also low. This cost measure is somewhat similar to our approach, however we find a specific preference value, similar to the inefficiency ratio, for each considered feature, where Letchner et al. only apply their ratio to the travel time. Another distinction is that their ratio if found using simple averaging over all observed GPS records for a single driver, whereas we utilize machine learning to find suitable parameters.

Dai et al. [5] recommend personalized routes by examining big trajectory data (more than 50,000 taxis from Beijing). They use Hidden Markov Models to learn individual preference vectors to define users driving preferences, like the framework presented in this paper also learns personalized preference vectors. However, they only consider three different preference features: travel time, distance, and fuel consumption, whereas our framework takes 12 features into account. To determine the probability of a user driving on a distinct edge, the authors implemented the PageRank method on a dual graph representation of the road network, somewhat similar to the line graph representation described in Section 3.3.

Balteanu et al. [6] explain how to compute shortest paths in OSM with respect to a specific preference distribution. They model these preferences as pairwise trade-offs between different cost factors, e.g. path distance vs. number of traffic signals. Using skyline routes (Pareto-optimal routes) they can derive the personal optimal path between two points in the road-network when given a personal preference gradient for the trade-off pair. Balteanu et al. only describe their proposed framework handling one trade-off pair (two cost factors) at one time as their main contribution was instead to efficiently find the skyline routes needed by the method. They also had to resort to experimenting with synthetic preference distributions as their trajectory data was anonymized. In contrast, we will be examining multiple cost factors instead of only two. We will also be testing the real-world applicability of our framework by using personal trajectory data in addition to synthetic data.

## 2.2 Context-Based Navigation

Yang et al. [7] examines context-aware personalized routing, where the context, e.g. travel distance, travel time and fuel consumption, of each trajectory is considered. Like [6], their work is also based on skyline routes. Yang et al. present a weighed Jaccard similarity for comparison of routes based on the distance of a route, whereas we utilize a more general Jaccard distance that only compares sets of road segments. Yang et al. use a data-first approach to identify the contexts considered by each driver by clustering trips with similar efficiency ratios: a concept capturing the importance of each feature with respect to the feature-optimal route. We also cluster trips, but based on their apparent preference towards road features when compared to alternative routes with the same start and end location. In contrast, we predefine what constitutes a context in order to use it for predicting the preference for unknown trips.

# 3 Modeling Road Networks

In this section, the modeling of road networks, trips, and features is explained. An overview over key notations can be seen in Table 1. Section 3.1 describes potential features in road networks. Section 3.2 explains how to model the road network using a graph structure with features on vertices, arcs, and pairs of consecutive arcs. Section 3.3 details how the graph structure is simplified using a line graph in order to only have features on graph arcs.

| Sym. | Description |
|---|---|
| $v$ | Vertex |
| $a$ | Arc: $a = (v_1, v_2)$ |
| $t$ | Turn: $t = (a_1, a_2)$ |
| $r$ | Route: $r = (t_1, t_2, \cdots)$ |
| $r.s$ | Start arc of a route: $r = ((r.s, a), \cdots)$ |
| $r.e$ | End arc of a route: $r = (\cdots, (a, r.e))$ |
| $\tau$ | Trip: $\tau = (r, \text{context})$ |
| $f$ | Feature function |
| $\boldsymbol{u}_t$ | Feature vector of turn $t$ |
| $\boldsymbol{x}$ | Feature vector of route |
| $\boldsymbol{X}$ | Feature matrix for multiple routes |
| $\boldsymbol{y}$ | Actual feature vector |
| $\boldsymbol{Y}$ | Actual feature matrix for multiple routes |
| $\boldsymbol{\beta}$ | Preference vector |
| $\boldsymbol{\beta}_\tau$ | Preference vector for $\tau$ |
| $\mathbf{0}_{ij}$ | Zero matrix with dimensions $i \times j$ |
| $\boldsymbol{\gamma}$ | Cluster assignment vector |
| $L$ | Loss function |

Table 1: Notation.

## 3.1 Road Network Features

Several features in road networks influence how people navigate them. Some features describe intersections (e.g. traffic signals). Other features describe road segments (e.g. length and road-type) or pairs of consecutive road segments (e.g. turns and transitions between road-types). Some of these features are static (e.g. segment length and road type), while other features are dynamic and changes depending on some context (e.g. segment duration depends on the time of day).

## 3.2 Graph Representation

In order to model all the different features concerning road networks we use the notion of a directed graph $G = (V, A)$, consisting of vertices $V$ and arcs $A$ where $A = \{(v_i, v_j) \mid v_i, v_j \in V\}$. Each vertex $v \in V$ describes either a shared point between two road segments (e.g. an intersection) or a dead end of a road segment. Each arc $a \in A$ describes a directional road segment, i.e. a one-directional road segment is represented by a single directed arc in the graph and a bi-directional road segment is represented by two directed arcs in opposite direction of each other. The vertices of $G$ can also contain all the intersection features of the road network, while the arcs can contain all the segment features. However, two problems arise in this graph model: First, features covering consecutive arcs cannot be represented directly in the graph structure but requires a separate lookup table. Second, path-finding algorithms must be modified to work with graphs containing features, and thus weights, on more than just the arcs. To get around these problems we instead use the line graph representation of the graph defined by Winter [8, 9] which allows for all the features to be placed on the arcs in the graph structure.

### 3.3 Line Graph Representation

The directed line graph $L(G) = (A, T)$ of a directed graph $G$ consists of arcs $A$ and *turns* $T$ defined as $T = \{(a_{ij}, a_{jk}) \mid a_{ij} \in A \wedge a_{jk} \in A\}$. The line graph $L(G)$ models transitions between road segments through a shared intersection. Each turn $t \in T$ from $L(G)$ thus represents a length-two path in $G$. An example of a directed graph and the corresponding directed line graph can be seen in Figure 2.

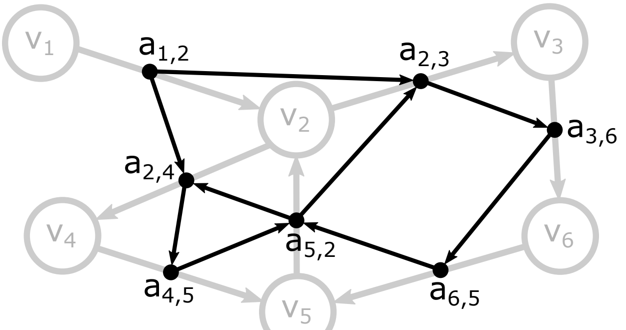


Figure 2: Example of a directed graph $G$ (grey) of vertices and arcs, and its directed line graph $L(G)$ (black) of arcs and turns.

The purpose of using a line graph of a road network is to model all road features (features of intersections, road segments, and pairs of consecutive road segments) using only turns in the line graph. Each turn $t_{ijk}$ connecting the arc $a_{ij}$ to $a_{jk}$ through the vertex $v_j$ contains the features of the arc $a_{ij}$, the vertex $v_j$, and the features of the consecutive arc pair $(a_{ij}, a_{jk})$. A small illustration of a single turn is seen in Figure 3.

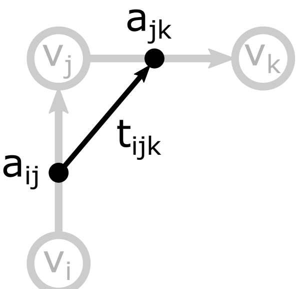


Figure 3: Example of a single turn $t_{ijk}$.

With a line graph representation of the road network, we can define a route $r$ to be sequence of consecutive turns $r = (t_1, t_2, \cdots, t_n)$ from $L(G)$. As an effect of this redefinition, the features of the last road segment in the trip are not considered, as the last turn $t_n$ only contains the features of the second to last road segment ($a_{ij}$ in Figure 3), the second to last intersection ($n_j$ in Figure 3), and the turn onto the last road segment ($t_{ijk}$ in Figure 3). However, ignoring the last road segment does not influence the model as all possible routes must include the last road segment, and with the preference towards all features assumed linear, the exclusion of the last road segment's features has the same impact on all routes ending in that segment.

Each turn $t$ contains a set of features which values we will represent using a *feature vector* $\boldsymbol{u}_t = (f_1, f_2, \cdots, f_n)$ where $n$ is the number of features in the road network and $f_i$ is the value of the $i$'th feature or 0 if the feature is not present in the turn. To hold all the features in the feature vector, they must all be represented using only numerical values.

## 4 Personalization

In this section, concepts and methods related to personalized preference are explained. The overall architecture of our framework can be seen in Figure 4. The framework is composed of two major components: **preference inference**, in which we find the preference vectors for each trip, and **preference modeling**, in which we build a model to predict the preference vector based on the context of the trip.

In Section 4.1 we formalize the concept and notation of personal preference. Section 4.2 lays the foundation for preference inference and Section 4.3 presents a performant method based on clustering. Section 4.4 briefly describes how we normalize the route features to overcome feature imbalance. Section 4.5 describes methods used to initialize the clusters. Section 4.6 explains how we implemented preference modeling using a neural network classifier.

### 4.1 Personal Preference

Personalization is achieved by assigning a personal preference towards each road feature. Using these preferences, the costs of each turn in the line graph can be calculated, making the shortest route between two road segments a personal optimal path according to the preferences. In other words, using the road features and their preference reduces the personalization routing problem to the shortest path problem. The only remaining problem then becomes learning the personal preferences associated with each driver. Furthermore, our previous results in [1] shows that every driver has several different sets of preferences that explain their behavior, extending the problem to learning all the different preferences and choosing the right one when planning future trips.

We model personal preference for $d$ features using a preference vector $\boldsymbol{\beta}$ of size $d$ where $\boldsymbol{\beta}_i$ is the preference towards the $i$'th feature. The cost of a feature vector $\boldsymbol{x}$ is defined as the sum of its elements each multiplied with the corresponding preference from $\boldsymbol{\beta}$:

$$\textsc{Cost}(\boldsymbol{x}, \boldsymbol{\beta}) = \boldsymbol{x} \cdot \boldsymbol{\beta} \tag{1}$$

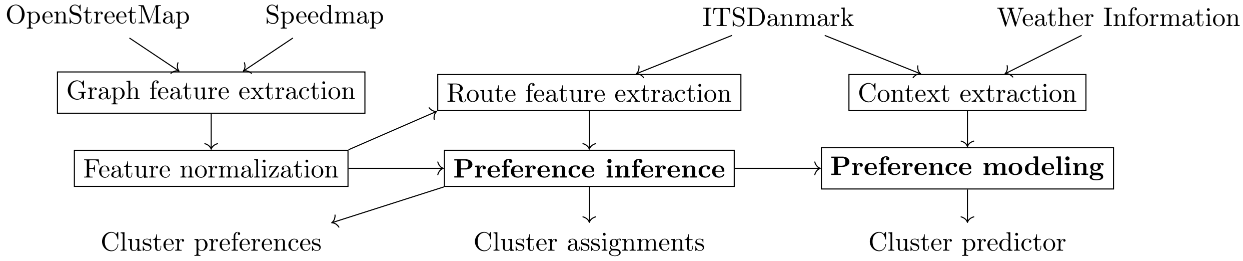


Figure 4: Framework overview. Arrows represent flow of data and rectangles represent processes. Inputs are located in top and outputs are located in the bottom.

The strictly linear definition of the cost function allows for easy aggregation of feature vectors as $\textsc{Cost}(\boldsymbol{x}_1, \boldsymbol{\beta}) + \textsc{Cost}(\boldsymbol{x}_2, \boldsymbol{\beta}) = \textsc{Cost}(\boldsymbol{x}_1 + \boldsymbol{x}_2, \boldsymbol{\beta})$, allowing the calculation of the cost of a single turn, sequences of turns, and whole trips.

## 4.2 Learning Preferences

A trip $\tau$ is defined as the tuple $(r, \text{context})$, where $r$ is the route driven and the context defines information about the driver, time, and weather. $\tau.r$ is a member of the set of routes that connects the arc $r.s$ to the arc $r.e$ defined as $R_e^s = \{(t_1, t_2, \cdots, t_n) \mid t_i \in T \wedge t_1 = s \wedge t_n = e \wedge t_i = (a, b) \wedge t_{i+1} = (b, c)\}$, where $T$ is the set of turns in the line graph $L(G)$. A preference vector $\boldsymbol{\beta}_\tau$ that exactly explains the trip $\tau$ from $s$ to $e$ is a preference vector such that $\forall r \in R_e^s(Cost(\boldsymbol{x}_r, \boldsymbol{\beta}_\tau) \geq Cost(\boldsymbol{x}_\tau, \boldsymbol{\beta}_\tau))$, where $\boldsymbol{x}_r$ is the feature vector for $r$ and $\boldsymbol{x}_\tau$ is the feature vector for $\tau$. $\boldsymbol{x}_r$ is defined as $\boldsymbol{x}_r = \sum_{i=1}^{n} \boldsymbol{u}_i$, where $n$ is the number of turns in $r$ and $\boldsymbol{u}_i$ is the feature vector for turn $i$.

In [3], this is seen as a linear programming problem. Each alternative route in $R_e^s \setminus \{\tau.r\}$ is seen as a linear constraint and the final preference vector is then any point in the feasible region. In the case where there is no feasible region (i.e. where there is no possible preference vector that makes the taken route optimal), the route is broken down into smaller sub-routes that are more likely to be feasible.

Instead of using these hard constraints as stated in the linear programming problem, we will use soft constraints and see it as an optimization problem and solve it using gradient descent. Soft constraints allows us to find an approximate solution when no exact solution exists.

For each alternative route, we define the loss $L$ as:

$$L(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{\beta}) = \textsc{Clamp}\left(0, \frac{\boldsymbol{y} \cdot \boldsymbol{\beta}}{\boldsymbol{x} \cdot \boldsymbol{\beta}} - 1, 1\right) \tag{2}$$

where $\textsc{Clamp}(a, b, c) = \max(a, \min(b, c))$, $\boldsymbol{y}$ is the features of the actual trip, $\boldsymbol{x}$ is the features of the alternative route for the trip, $\boldsymbol{\beta}$ is the preference vector of the route.

$d = \frac{\boldsymbol{y} \cdot \boldsymbol{\beta}}{\boldsymbol{x} \cdot \boldsymbol{\beta}} - 1$ is the proportional cost of alternative route compared to the actual route. We wish to minimize this, i.e. the actual route should become cheaper and the alternative route should become more expensive. We clamp $d$ between 0 and 1 for two reasons:

- For $d < 0$, the actual route is cheaper than the alternative route and there is no benefit in improving it further.
- For $d > 1$, the alternative route is half or less the cost of the actual route. We limit the loss $L$ in this case, as otherwise a few wildly inefficient trips will dominate the total across all trips. For example, if a driver is delivering a package, but not stopping long enough for the trip to be split into two separate trips.

A problem with using gradient descent to minimize the loss function $L$ is that it can result in negative weights in the preference vector. Having features that are weighted negatively can create arcs with negative costs, which in turn can lead to negative cycles. Negative weights are also incompatible with many shortest-path algorithms such as Dijkstra's algorithm. To prevent these negative weights, we take the absolute value of $\beta$ during optimization and pathfinding. Strictly, this does not prevent negative values in $\beta$, but it does ensure that it will always function as if it was positive. We define ABS as a function returning the absolute value for each scalar:

$$\textsc{abs}((v_1, v_2, \cdots, v_n)) = (|v_1|, |v_2|, \cdots, |v_n|) \tag{3}$$

Solving this optimization problem would necessitate enumerating all of the routes between $r.s$ and $r.e$, of which there are exponentially many for a single pair. Similar to [1, 3], we solve this by starting from an initial alternative route and then iteratively solving the optimization problem, using the new preference vector to find a new alternative route, and adding the new alternate route to the constraints. We will refer to this as a *pathfinding step.*

To solve the optimization problem, we use a gradient descent method that iteratively searches for a local optimum. We will refer to one such iteration as an *optimization step.* This method will eventually stabilize around a (possibly local) optimum, in which case we need to perform the pathfinding step again. This will change the location of the optimum, and we then perform further optimization steps. We call the number of optimization steps per pathfinding step $T_{\text{pathfinding}}$. The entire algorithm can be seen in Algorithm 1.

**Algorithm 1** Preference estimation of a single route.

```
Require: $r$ is a route, $G(A,T)$ is a line graph, $f$ is a feature function $T \to \mathbb{R}^d_+$ where $d$ is the number of features, $n_{\text{iterations}}$ is the number of iterations, $T_{\text{pathfinding}}$ is the number of iterations per pathfinding
function EstimateSinglePreference($r, G, f$)
    $m \leftarrow \left\lceil \frac{n_{\text{iterations}}}{T_{\text{clustering}}} \right\rceil$        ▷ $m$ is the number of times we will find alternative routes
    $h \leftarrow 0$        ▷ $h$ is the number of alternative routes found so far
    $\boldsymbol{\beta} \leftarrow$ InitialPreferences($d$)        ▷ $\boldsymbol{\beta}_i$ is the $i$'th feature of the preference vector
    $\boldsymbol{X} \leftarrow \mathbf{0}_{md}$        ▷ $\boldsymbol{X}_{ij}$ is the $j$'th feature of the $i$'th alternative route
    $\boldsymbol{y} \leftarrow \sum_{t \in r} f(t)$        ▷ $\boldsymbol{y}_i$ is the $i$'th feature of route $r$
    for $i \leftarrow 1, n_{\text{iterations}}$ do
        if $i \bmod T_{\text{pathfinding}} = 1$ then        ▷ Pathfinding step
            $h \leftarrow h + 1$
            $p \leftarrow$ ShortestPath($G, r.s, r.e, g$) where $g(t) = f(t) \cdot \text{ABS}(\boldsymbol{\beta})$
            $\boldsymbol{X}_h \leftarrow \sum_{t \in p} f(t)$
        end if
        $\boldsymbol{\beta} \leftarrow \text{Optimize}\left( \sum_{j=1}^{h} L\left(\boldsymbol{y}, \boldsymbol{X}_j, \text{ABS}(\boldsymbol{\beta})\right) \right)$        ▷ Optimization step
    end for
    return ABS($\boldsymbol{\beta}$)
end function
```

**Algorithm 2** Clustered preference estimation over multiple routes.

```
Require: $R$ is a set of routes, $G(A,T)$ is a line graph, $f$ is a feature function $T \to \mathbb{R}^d_+$ where $d$ is the number of features, $n_{\text{iterations}}$ is the number of iterations, $n_{\text{clusters}}$ is the number of clusters, $T_{\text{pathfinding}}$ is the number of iterations per pathfinding, and $T_{\text{clustering}}$ is the number of iterations per clustering
function EstimateMultiPreferences($R, G, f$)
    $m \leftarrow \left\lceil \frac{n_{\text{iterations}}}{T_{\text{clustering}}} \right\rceil$        ▷ $m$ is the number of times we will find alternative routes
    $h \leftarrow 0$        ▷ $h$ is the number of alternative routes found for each route so far
    $\boldsymbol{\beta} \leftarrow$ InitialPreferences($n_{\text{clusters}}, d$)        ▷ $\boldsymbol{\beta}_{ij}$ is the $j$'th feature of the $i$'th preference vector
    $\boldsymbol{X} \leftarrow \mathbf{0}_{|R|md}$        ▷ $\boldsymbol{X}_{ijk}$ is the $k$'th feature of the $j$'th alternative route of the $i$'th actual route
    for $i \leftarrow 1, |R|$ do
        $\boldsymbol{Y}_i \leftarrow \sum_{t \in R_i} f(t)$        ▷ $\boldsymbol{Y}_{ij}$ is the $j$'th feature of the $i$'th route
    end for
    $\boldsymbol{\gamma} \leftarrow$ InitialClusters($|R|$)        ▷ $\boldsymbol{\gamma}_i$ is the cluster that trip $i$ is currently assigned to
    for $i \leftarrow 1, n_{\text{iterations}}$ do
        if $i \bmod T_{\text{clustering}} = 1 \wedge i \neq 1$ then        ▷ Cluster assignment step
            for $j \leftarrow 1, |R|$ do
                $\boldsymbol{\gamma}_j \leftarrow \operatorname*{arg\,min}_{k=1}^{n_{\text{clusters}}} \sum_{l=1}^{h} L(\boldsymbol{Y}_j, \boldsymbol{X}_{jl}, \boldsymbol{\beta}_k)$
            end for
        end if
        if $i \bmod T_{\text{pathfinding}} = 1$ then        ▷ Pathfinding step
            $h \leftarrow h + 1$
            for $j \leftarrow 1, |R|$ do
                $p \leftarrow$ ShortestPath($G, R_j.s, R_j.e, g$) where $g(t) = f(t) \cdot \text{ABS}(\boldsymbol{\beta}_{\boldsymbol{\gamma}_j})$
                $\boldsymbol{X}_{jh} \leftarrow \sum_{t \in p} f(t)$
            end for
        end if
        $\boldsymbol{\beta} \leftarrow \text{Optimize}\left( \sum_{j=1}^{|R|} \sum_{k=1}^{h} L(\boldsymbol{Y}_j, \boldsymbol{X}_{jk}, \text{ABS}(\boldsymbol{\beta}_{\boldsymbol{\gamma}_j})) \right)$        ▷ Optimization step
    end for
    return ABS($\boldsymbol{\beta}$), $\boldsymbol{\gamma}$
end function
```

## 4.3 Clustering

Each preference vector to be learned requires its own instance of the road network graph as it is needed to sample alternative routes from $R_e^s$ using that preference. Learning a preference for each trip would be an unnecessarily complex and memory consuming task and we also hypothesize that all the trips can be described by a much smaller set of preferences. In [1] a preference is found for each driver, and used to predict all their future trips. Here we will instead try to identify a set of preferences that each trip will be assigned to, making preferences more general but allowing each driver to have multiple possible preferences to choose from.

To learn which trips share preferences, we introduce clustering in our training process. For a predetermined number of clusters $k$, we want to learn $k$ preference vectors, each representing a cluster, and assign each trip $\tau$ to the cluster preference that best explains it.

The most intuitive method of performing this clustering would be in two steps:

– Find preferences for each trip.

– Perform clustering over the preferences.

However, this requires computing a preference vector for each trip, only to reduce it to fewer clusters. We propose an extension of Algorithm 1 using a method similar to the expectation-maximization (EM) algorithm [10] in order to perform clustering without first computing individual preference vectors, thus avoiding the expensive step of computing an individual preference vector for each trip.

The algorithm works by first assigning each trip to a cluster. We a priori decide $m$, the number of clusters we wish to find among the $n$ trips with $d$ features. Then we initialize the $m \times d$ preference matrix $\boldsymbol{\beta}$, and the $n$-vector $\boldsymbol{\gamma}$, containing the currently assigned cluster of each trip. We explore different ways of initializing these values in Section 4.5.

In the expectation step, each trip is assigned the cluster that minimizes the loss $L$ of its alternative routes. We will refer to this as the *cluster assignment step.*

In the maximization step each cluster's preference is updated based on the trips assigned to it. Each cluster will optimize its preference vector by minimizing the loss using all its assigned trips over a fixed amount of iterations. This is basically the optimization step in Algorithm 1, but extended to several clusters.

The expectation and maximization steps are then executed in a loop until the cluster preferences no longer change significantly. This process is analogous to other EM algorithms such as K-Means clustering, where we in one step move the cluster centroids (maximization), and in another assign points to the nearest cluster (expectation). However, in our case, the maximization step requires several iterations due to gradient descent. We will refer to the number of iterations per cluster assignment as $T_{\text{clustering}}$.

The pseudocode of the algorithm can be seen in Algorithm 2.

## 4.4 Feature Normalization

The route features in our graph have an extremely high variance between them. Meters driven are often in the tens of thousands, while bridges crossed are usually 0 or 1. This high variance in our dataset means the gradient descent step in the *meters driven* direction will have a far larger impact than in the *bridges crossed* direction. To overcome this imbalance, we normalize the features such that each feature on each turn has a average value of 128.

We normalize to 128 instead of 1 due to the pathfinding implementation we use: RoutingKit. RoutingKit only accepts integer costs, so normalizing to 1 would mean a large loss of precision to rounding, whereas normalizing to 128 leaves 7 bits to decimal values. Theoretically, a larger value would be even more precise, but would also have additional risk of integer overflow during pathfinding.

## 4.5 Initial Preferences

The algorithms described in Section 4.2 and Section 4.3 requires a method to initialize the preference vectors (and in the case of Algorithm 2, to initialize the cluster assignment). The primary concern when initializing these values, is to ensure that the clusters do not become empty during the assignment step. An empty cluster would prevent the cluster from improving in the maximization step, as there is no loss for the optimizer to improve, essentially making it useless. We consider two different methods of initialization:

– Random initialization: each cluster is randomly initialized.

– Common initialization: all clusters are initialized to the same value.

Random initialization has the quality that very few assumptions are made. However, one problem is that only few of the generated preference vectors are likely to be realistic, making it more likely to be empty during cluster assignment.

In the common initialization method, we initialize the preference vectors to a common starting point, e.g. shortest-time or shortest-distance. We assume people are primarily interested in shortest-time or shortest-distance, which would make this a good initial preference.

However, this presents another problem: the ABS function in equation 3 has no derivative in 0. Thus, initializing features to 0 will mean there is no gradient for those features which would lead to them never changing. Instead, we therefore initialize all features in the preference vector to 1.

Likewise, it is also necessary to initialize the cluster assignment. One option is to perform the cluster assignment immediately, but this leaves the possibility of a cluster not receiving any trips. A better option is to assign each trip to a random cluster, perform maximization, and then only afterwards perform cluster assignment. This makes it much more likely that at least one trip is assigned to the cluster, as it is now optimized to the randomly assigned initial trips.

### 4.6 Predicting Preferences

In order to know which preference to use for predicting the route of future trips, a new model can be trained with the context of each trip and its assigned cluster preference. This model should be able to predict the preference to use for routing, given the context of the trip. Table 2 shows the identified contexts for trips and how they are encoded.

| Context | Encoding |
|---|---|
| Driver id | One-hot |
| Start time of day | Circular |
| Start day of week | One-hot |
| Start season | One-hot |
| Start weather | One-hot |
| Start location | One-hot ranking |
| End location | One-hot ranking |

Table 2: List of contexts and their corresponding encoding method.

In a one-hot encoding, all possible discrete values of a context (e.g. *rain*, *snow*, *fog*, etc. for start weather) are each encoded as a feature. The feature which name matches the input value takes the value 1 while all other encoded features of the context have the value 0.

The start time of day is encoded using a 2-dimensional 24-hour circular "clock-face" where a point in time is represented as a spatial point $(x, y)$. This encoding ensures that points close in time always lie close in the encoded space (e.g. 23:59 is close to 00:00) unlike in a simple continuous encoding.

The start and end location features are based on a ranking system. Each user has their trips' start road segments grouped and ordered by number of occurrences. The five most frequent starting segments are transformed to one-hot encoded features and the same process is used for the end segments. Thus, the start and end location features of a driver wanting to navigate from their most frequent start location to their third most frequent end location would be $(1, 0, 0, 0, 0)$ and $(0, 0, 1, 0, 0)$ respectively, while a location the driver does not frequent is encoded as $(0, 0, 0, 0, 0)$.

The model chosen for predicting the cluster preference is a neural network with one input layer, a single hidden layer with dropout regularization and RELU activation, and an output layer with softmax activation. The input layer is given the values of the encoded contexts for all known trips, and the network is learned to minimize either the loss, $L$, or squared loss, $L^2$, of the trips by choosing an appropriate soft assignment of cluster preferences.

The encoded contexts listed in Table 2 results in 491 individual input values for the neural network, while the number of output values equals the number of clusters decided on prior to the clustering step.

The hyperparameters of the model are: loss function choice (default, squared), learning rate ($\alpha$), dropout regularization percentage, and the number of neurons in the hidden layer. Section 5.7 explores how to find optimal values for the hyperparameters.

## 5 Experiments

All experiments were conducted on a computer with two Intel Xeon Quad-Core E5620 processors (2.40 GHz, 4 Cores, 8 Threads, 12M Cache), 48 GB DDR3 RAM (1333 MHz), and a NVIDIA GeForce GTX 1070 OC (1.822 GHz core clock rate, 1,920 CUDA cores, and 8 GB of GDDR5 memory), running Ubuntu 16.04, PostgreSQL 9.6, PostGIS 2.3.2, and TensorFlow 1.0.1. For handling the data on disks we utilized file system compression (explained in detail in Appendix A).

### 5.1 Data Foundation

The dataset used in this paper consists of 1.306 billion map matched GPS records from the ITS (Intelligent Transport Systems) project[1] gathered during a period between 2012 and 2014 (964 days), primarily in North Jutland, Denmark. The GPS records have been map matched to 1.381 million trips from 458 drivers. The trips are matched to the OpenStreetMap (OSM) road network[2]. Preliminary data analysis can be seen in Appendix C.

Additional data includes weather information for the North Jutland in the time period 2012 to 2014 and a speedmap containing the average driving speed profile of the road segments in North Jutland, both provided by Aalborg University. This speedmap is based on GPS records of taxi drivers and forms the basis for our dynamic feature: *duration*. An average speed is provided for each 15 minute interval in a day and has been calculated by averaging speed measurements on road segments during the weekdays. The speed for road segments and time interval combinations with no measurements is inferred from other measurements (see Appendix C.3).

Due to the speedmap only containing speeds for roads in North Jutland and not all of Denmark, we constrain the OSM road network used in this paper to only include the roads also included in the speedmap. After filtering away trips from the ITS dataset not within our constrained road network, 900 million records from 1.267 million trips remains from the same 458 drivers. The potential for personalization increases with the length and duration of a trip, as short trips rarely have significantly different route alternatives. Therefore, we filter out trips shorter than 500 meters and with a duration of less than 3 minutes (thresholds based on the analysis described in Appendix C.2). After filtering, the total number of trips is reduced from 1267 million

[1] itsdanmark.dk/Om-os/ITSDanmark-(English)

[2] planet.osm.org/planet/2014/planet-140101.osm.bz2

to 958 million (a reduction of 25.4 %). An overview of the filtering steps can be seen in Table 3.

| Filtering steps | GPS records | Trips | Trips/ driver |
|---|---|---|---|
| No filter | 1,306 M | 1,381 K | 3015 |
| +North Jutland | 900 M | 1,267 K | 2766 |
| +Length/duration | 866 M | 958 K | 2091 |

Table 3: Total count of GPS records, trips, and trips per driver after applying each filtering step given all 458 drivers over a period of 964 days.

| Feature | Unit |
|---|---|
| Duration | Seconds |
| Distance | Meters |
| Motorway | Meters |
| Rural | Meters |
| Other road type | Meters |
| Left turns | 1 |
| Right turns | 1 |
| U-turns | 1 |
| Traffic obstacles | 1 |
| Crossings | 1 |
| Bridges | 1 |
| Tunnels | 1 |

Table 4: Road features extracted from OSM and speedmap.

OSM contains static road features such as: distance, road type, traffic signals, bridges, and tunnels. All of the features used in this project are outlined in Table 4. We will not perform experiments to see what features are significant, as the framework is capable of determining it during training. The insignificant features stand out by being weighted low by all clusters. As an example of this, the *tunnel* feature proves to be very insignificant for our road network, as seen in Table 10 in Appendix D.

## 5.2 Shortest Path Computation

As Section 4 describes, the shortest path in a line graph $G$ becomes the personal optimum path when using a preference vector in combination with the road features to calculate the weights of $G$. The shortest path is used to generate alternative routes during preference estimation in Algorithm 1 line 10 and when new personalized routes should be found. The popular shortest path algorithms *Dijkstra* [11] and *bidirectional Dijkstra* are too slow for larger graphs the size of countries [2]. Faster methods exist, where the graph is preprocessed to allow for better performance during pathfinding [12, 13, 14].

In line with our previous work [1], we use *Customizable Contraction Hierarchies* (CCH) by Dibbelt et al. [13], an extension of the Contraction Hierarchy method by Geisberger et al. [14]. CCH consists of two steps: preprocessing and customization. The preprocessing step identifies cliques and shortcuts in the graph and only has to be performed once for the whole graph. The customization step updates the weights in the preprocessed graph and must be performed each time the preference vector is updated. The preference vector is constantly updating when computing the clusters in Algorithm 2, making the relatively slow customization step a computationally expensive part of the training process. However, the customization run time is bounded by the size of the graph, and does not depend on the number of trips, making it very scalable.

## 5.3 Clustering Hyperparameters

The model described in Section 4.2 depends on several different hyperparameters, such as learning rate and the optimization method used. To find the optimal hyperparameters, we perform grid search across the following hyperparameter dimensions:

- Number of clusters: The number of clusters is likely to have a significant effect on the quality of personalization. More clusters can potentially lead to better personalization, but also decreases performance.

- Optimizer: The choice of optimizer can also have a significant effect on model performance. We evaluate two optimizers: Gradient Descent (GD) and the Adam optimizer [15]. Compared to GD, the Adam optimizer generally performs better in neural networks but it is unknown if it performs better for route personalization.

- Learning rate: The learning rate can have a significant influence. Too low and the model will take a long time to converge. Too high and the model may never converge to a local minimum. In addition, we will also try an exponentially decaying learning rate (see Appendix B.1). Adam already does adaptive learning rate optimization [15], so decaying learning rate is generally less useful for Adam.

There are a number of other hyperparameters which we will not explore in the grid search, mostly due to the already high run time of our grid search. These hyperparameters include $n_{\text{iterations}}$, $T_{\text{pathfinding}}$, and $T_{\text{clustering}}$ which are set to 10,000, 1000, and 1000 respectively.

The result of our grid search can be seen in Table 5. We can see that the Adam optimizer generally performs better than GD. Adam also seems to generally perform well independently of the learning rate, with the exception of $\alpha = 10^{-4}$ which performs significantly worse. We believe this may be due to how Adam already implements $\alpha$-decay and the effective learning rate may therefore be too low to make any significant improvements in the case of $\alpha = 10^{-4}$. We can also see that the loss improves as more clusters are added, but with diminishing returns. Only having a single cluster (effectively not clustering), yields a significantly higher loss than more clusters.

|  | Method: | Adam Optimizer |  |  |  |  |  | Gradient Decent |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | Clusters: | 1 | 2 | 4 | 8 | 12 | 16 | 1 | 2 | 4 | 8 | 12 | 16 |
| $\alpha$ | $10^{-2}$ | 166 | 126 | 103 | 99 | 92 | 88 | 184 | 152 | 130 | 130 | 118 | 114 |
|  | $10^{-3}$ | 164 | 134 | 108 | 89 | 88 | **85** | 181 | 151 | 138 | 125 | 117 | 118 |
|  | $10^{-4}$ | 288 | 241 | 191 | 181 | 145 | 152 | 182 | 146 | 131 | 116 | 113 | 108 |
|  | $0.95^{i/100}$ | 163 | 118 | 104 | 91 | 86 | 88 | 165 | 146 | 132 | 121 | 114 | 100 |

Table 5: Percentile average loss across alternative routes (lower is better).

Each run took an average of 3.9 hours to complete. However, we did not have exclusive use of the hardware in the time frame the experiment was running, and the accuracy may therefore be doubtful.

## 5.4 Number of Trips

To gain an idea of how many trips are necessary to infer the preferences of a user, we conduct an experiment in which we choose a preference vector, generate several trips with that preference vector, and see how many trips are necessary for our preference inference method to infer the original preference vector from the generated routes.

The baseline preference vector is selected by performing preference inference over the entire dataset without clustering. The result is a preference vector that generalizes all trips. We then generate $n$ routes using that baseline vector and a random selection of start and endpoints from our dataset. We then perform single cluster preference inference with these generated routes as input and compare the estimated preference to the baseline. By varying this $n$, we can see how well the system performs under different number of trips. Cosine distance is used to compare the vectors, as this is a scale invariant distance measure.

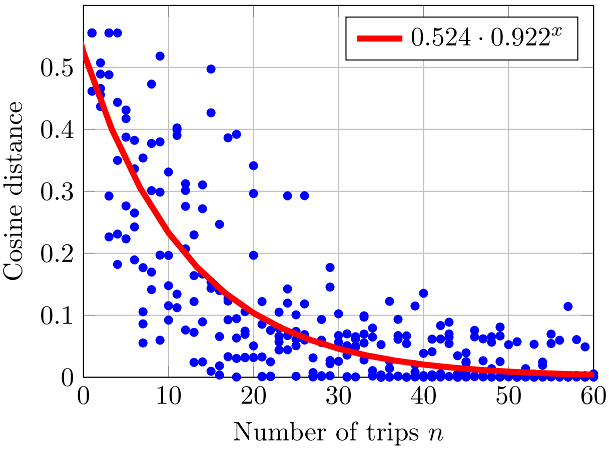


Figure 5: Cosine distance between inferred and true preference vectors (lower is better) as a function of number of trips.

The result can be seen in Figure 5. We can see that as the number of trips increase, the cosine distance decreases. The trendline shows a exponentially decaying relationship between the number of trips and the cosine distance. For this particular preference, we can see that approximately 40 trips are required to approximate the original preference well, however the exact relationship may vary between different preference vectors.

## 5.5 Synthetic Preference Inference

To determine the efficacy of clustering, we conduct an experiment in which we generate a synthetic dataset based on a set of known preferences and see if these preferences can be inferred from the generated routes.

Using the method described in Section 3 and Section 4 we learn 16 cluster preferences for the dataset presented in Section 5.1. Our 958 K real trips are then randomly distributed between the 16 preferences and a synthetic route is generated for each trip based on the assigned preference. This assigned preference is regarded as the ground truth for the generated trips. Using all the generated trips of a preference, the model from Section 3 and Section 4 is then used again to re-learn the preference that generated them.

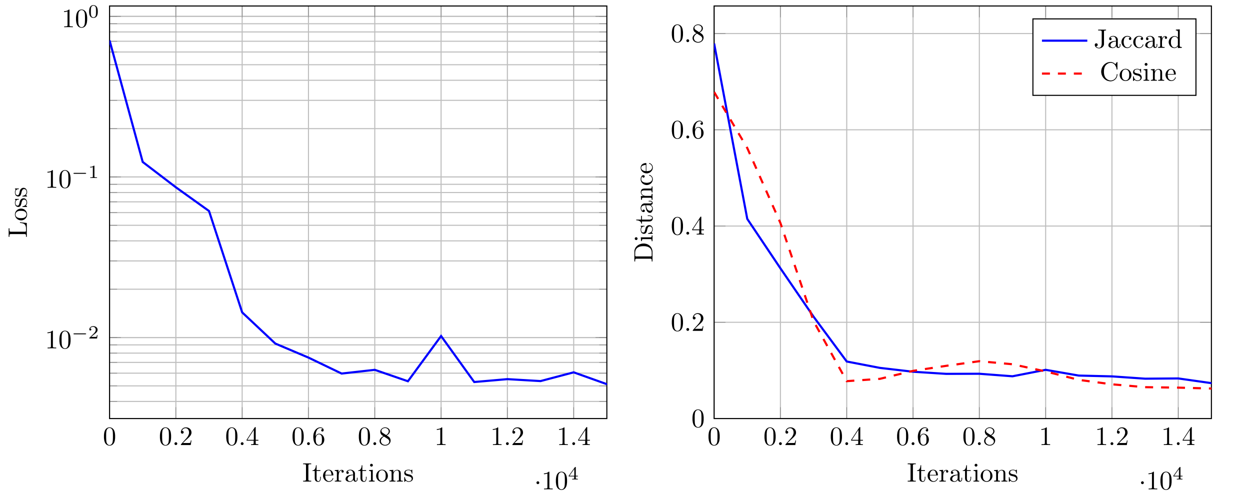
As in Section 5.4, we use cosine distance to compare a re-learned preference vector to its true preference vector. However, we cannot expect this experiment to reach a perfect similarity, because a route may have many equivalent preference vectors. These preference vectors can be quite different, while still resulting in the exact same route.

In addition to comparing the inferred preference vectors, we will also compare the similarity of the routes. For this purpose, we consider a route as a set of turns and the distance between them as the Jaccard distance of the two sets. The Jaccard distance of two sets $A$ and $B$ is defined as:

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{4}$$

The results of this experiment is presented in Figure 6b and Figure 6a. In Figure 6a, we can see that the loss declines exponentially before stabilizing around $5.5 \cdot 10^{-3}$.

In Figure 6b, we can see that while the cosine distance does decrease initially, it does not approach 0 but stabilizes around 0.07. This is likely due to trips having many different equivalent preferences. In the previous experiment with only a single cluster, this could be diminished by adding more trips. However, this experiment has more clusters, and thus each trip has some degrees of freedom in what cluster it can be assigned to. Adding more trips are therefore less likely



(a) Loss as a function of number of training iterations.

(b) Jaccard and cosine distance as a function of number of iterations.

Figure 6: Loss and distance metrics over 15,000 iterations for a synthetic dataset (lower is better).

to improve the overall cosine distance. We can also see that the Jaccard distance and cosine distance seem correlated.

## 5.6 Comparison with Real Trips

The previous experiments have been based mostly on synthetic data, due to the ease at which we can create a ground truth. To compare against real-world trips, we cannot use a metric like the cosine distance as it requires the true preference to be known. We therefore only use the Jaccard distance.

In this experiment, we start from our initial preferences from Section 4.5 and perform path inference using the hyperparameters from Section 5.3. To evaluate the inferred preferences, we compare the real route taken by a user to the one taken by the inferred preference vector.

| Preference | Jaccard Distance |
|---|---|
| Inferred | 0.482 |
| Fastest | 0.559 |
| Shortest | 0.967 |

Table 6: Avg. Jaccard distance using different preferences

The resulting distribution of Jaccard distances can be seen in Figure 8 and Table 6. We can see that the inferred route performs very similar to the fastest route. The inferred preference has more trips in the range $[0.0, 0.2)$ and is better on average. The shortest route, however, has a high Jaccard distance for nearly all trips.

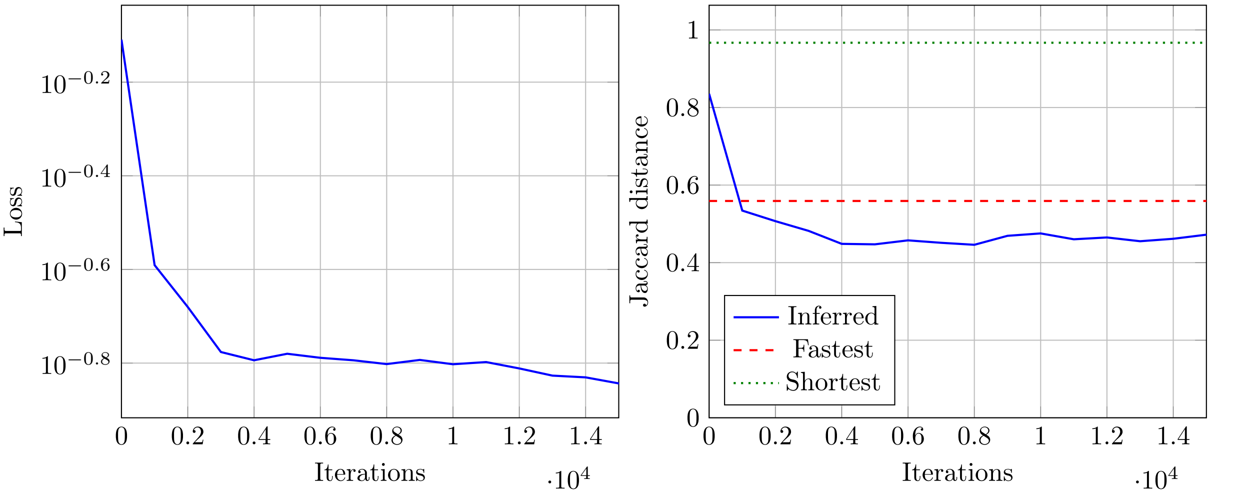
Compared to the synthetic experiment in Section 5.5, we start similarly at a very high Jaccard distance. However, the experiment on the real data seems to stagnate just below 0.5, whereas the synthetic experiment continues downward. The final Jaccard distance of 0.48 is significantly higher than the synthetic Jaccard distance of 0.09. We can interpret this in a number of ways:

- The inferred route is feature-wise similar, but not necessarily an overlapping route. A non-overlapping route may still be a good approximation of the true preferences of a user, and thus the preference may still be a good approximation.
- The user preferences are non-linear. A basic assumption of our framework is that the user preferences are linear. If this is not the case, then we cannot expect the system to find a suitable preference vector.
- The user preferences may depend on features that are not modeled in our graph. This may be due to the features not being present in our source data (e.g. road work).

In Section 5.5, we found the Jaccard distance followed the cosine distance closely, which lends more credence to the two latter interpretations. It is highly likely that there are features not being modeled that users consider, but we would expect duration and distance to be the most significant features. Duration is highly dynamic, often depending on local events, accidents, roadwork, etc., and it may therefore be that our duration feature is inaccurate.

The synthetic routes always follow a preference exactly. This might not be the case for real drivers, as they are unlikely to have perfect information of all road features. Especially if a driver is driving in an area they are unfamiliar in, it would be more likely for them to take a route that is suboptimal with respect to their actual preferences, e.g. the driver might not have sufficient information about the duration of different route alternatives.

All of these effects may influence the inferred preference, and may be why we do not see a significant improvement lower than 0.5. However, it does perform



(a) Loss as a function of number of training iterations.

(b) Jaccard distance as a function of number of iterations for inferred and baseline preferences.

Figure 7: Loss and distance metrics over 15,000 iterations for the real dataset (lower is better).

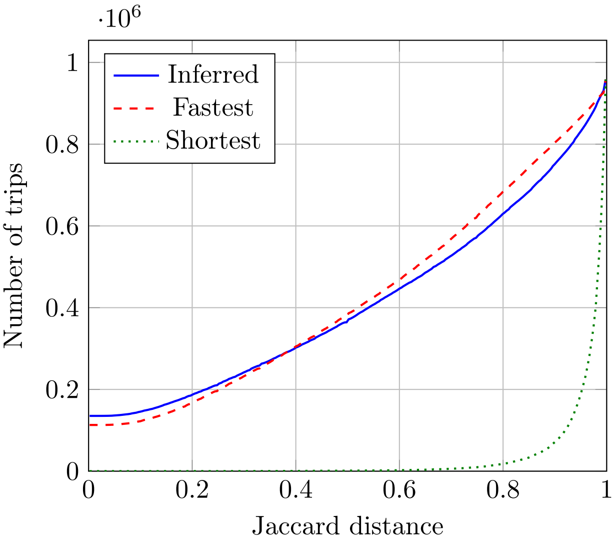


Figure 8: Number of trips with Jaccard distance less than $x$ for different preferences

slightly better than the fastest route, and considerably better than the shortest route.

## 5.7 Classification Hyperparameters

As with the hyperparameter exploration discussed in Section 5.3, this section presents the hyperparameter search results for the classification method described in Section 4.6.

The parameters we explore are:

- Loss function: We can either use the *default* loss described in Equation 2 or use a *squared loss* to penalize wrong decisions even more.
- Learning rate: As described in Section 5.3 the learning rate can greatly influence the final accuracy of the learned model.
- Dropout percentage: Dropout is a regularization method used to avoid overfitting. The dropout percentage decides how many neurons are randomly frozen during each training step. Appendix B.2 further explains dropout regularization.
- Hidden neurons: The number of hidden neurons in the model affects the complexity of the model. Fewer neurons results in more general, but sometimes inaccurate, models. Many neurons allows complex relationships to be modeled, but is more prone to overfitting and generally requires more training data and training time.

To find the optimal parameters for the model, we perform a grid search across the four dimensions described above. Using Adam as the optimization method, 90 % of the data for training, and 5 % of the data for evaluation we obtain the results shown in Table 7. The remaining 5 % is used for validating our results. We also compare the results to a baseline that randomly decides on a cluster preference for each trip in the evaluation set.

The parameter with highest influence on the results is the learning rate. As each configuration only encounters each training instance once, a lower learning rate with a slower approach towards the optimum might not reach it over the course of a single training epoch. Dropout does not have a large impact, except in configurations with lower learning rates.

We validate the results using the remaining 5 % of the data that was not used for training or testing. We only validate the percentile average loss of the two highlighted configurations in Table 7. For the configuration with 32 hidden neurons the percentile average loss was 83.3 and with 256 hidden neurons it was 83.1. These loss values are optimal as the theoretical lowest loss is 85 according to Table 5. The reason for obtaining a lower loss is due to the loss listed in Table 5 being calculated before the final re-clustering step, which could lower the loss further.

|  | Loss | Default |  |  |  | Squared |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
|  | Neurons | 32 | 64 | 128 | 256 | 32 | 64 | 128 | 256 |
| Dropout | $\alpha$ |  |  |  |  |  |  |  |  |
| 0 % | $10^{-2}$ | **81.0** | 81.3 | 81.1 | **81.0** | 81.2 | 81.6 | 82.5 | 82.5 |
|  | $10^{-3}$ | 89.3 | 84.1 | 83.0 | 84.0 | 83.8 | 83.0 | 83.3 | 82.8 |
|  | $10^{-4}$ | 119.2 | 90.0 | 94.0 | 89.7 | 108.4 | 93.6 | 99.9 | 92.1 |
| 25 % | $10^{-2}$ | 81.8 | 81.7 | 84.0 | 82.5 | 81.7 | 82.5 | 81.7 | 81.4 |
|  | $10^{-3}$ | 85.0 | 84.0 | 83.0 | 82.6 | 85.9 | 84.2 | 84.3 | 84.9 |
|  | $10^{-4}$ | 93.4 | 104.4 | 106.8 | 100.0 | 103.1 | 99.4 | 101.5 | 96.2 |
| 50 % | $10^{-2}$ | 82.5 | 82.2 | 82.0 | 82.0 | 82.5 | 82.5 | 82.2 | 82.5 |
|  | $10^{-3}$ | 87.5 | 90.4 | 84.9 | 85.1 | 90.7 | 85.8 | 83.4 | 84.2 |
|  | $10^{-4}$ | 111.6 | 105.2 | 101.0 | 98.3 | 97.4 | 121.2 | 104.8 | 98.1 |
| Random baseline (average of 100 runs): 112.7 |  |  |  |  |  |  |  |  |  |

Table 7: Percentile average loss when predicting new routes (lower is better).

Each configuration took on average 6.27 minutes to complete, but run time could be decreased by training with larger batches at the price of an increased loss.

## 6 Conclusion

We present a novel method of modeling road networks to allow for personalized navigation, based on our previous work [1]. We detail how to identify different types of features in road networks and how to model them in a graph structure. We also extend our previous work to include context-based preference inference by using a combination of preference clustering and classification based on context. We define a loss function, $L$, that both the clustering and classification methods aim to minimize.

We identify optimal parameters for both preference inference and preference modeling using grid search and present the best configurations for our dataset.

Section 5.4 shows that around 40 trips are required to approximate a linear preference, and that the distance to the true preference is exponentially decaying in the number of trips used for training. Section 5.5 further demonstrates that a linear preference can be approximated to within a cosine distance of 0.07 and a Jaccard distance of 0.09. However, Section 5.6 demonstrates that this assumption only weakly holds for real-world users, as the Jaccard distance is much higher with values of 0.48 and 0.56 for the inferred route and the fastest route respectively.

## References

[1] Joachim Klokkervoll, Samuel Nygaard, Mike Pedersen, Lynge Poulsgaard, Philip Sørensen, and Henrik Ullerichs. Mining driving preferences and efficient personalized routing. *Det Digitale Projektbibliotek, Aalborg Universitet*, 2016. URL http://projekter.aau.dk/projekter/da/studentthesis/mining-driving-preferences-and-efficient-personalized-routing(1f825b85-9db8-4fbe-8753-aa117812ddd8).html.

[2] Daniel Delling, Andrew V. Goldberg, Moises Goldszmidt, John Krumm, Kunal Talwar, and Renato F. Werneck. Navigation made personal: Inferring driving preferences from gps traces. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '15, pages 31:1–31:9, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3967-4. doi: 10.1145/2820783.2820808. URL http://doi.acm.org/10.1145/2820783.2820808.

[3] Stefan Funke, Sören Laue, and Sabine Storandt. Deducing individual driving preferences for user-aware navigation. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '16, pages 14:1–14:9, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4589-7. doi: 10.1145/2996913.2997004. URL http://doi.acm.org/10.1145/2996913.2997004.

[4] Julia Letchner, John Krumm, and Eric Horvitz. Trip router with individualized preferences (trip): Incorporating personalization into route planning. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1795. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[5] J. Dai, B. Yang, C. Guo, and Z. Ding. Personalized route recommendation using big trajectory data. In *2015 IEEE 31st International Conference on Data Engineering*, pages 543–554, April 2015. doi: 10.1109/ICDE.2015.7113313.

[6] Adrian Balteanu, Gregor Jossé, and Matthias Schubert. Mining driving preferences in multi-cost networks. In *Proceedings of the 13th International Conference on Advances in Spatial and Temporal

*Databases*, SSTD'13, pages 74–91, Berlin, Heidelberg, 2013. Springer-Verlag. ISBN 978-3-642-40234-0. doi: 10.1007/978-3-642-40235-7_5. URL http://dx.doi.org/10.1007/978-3-642-40235-7_5.

[7] Bin Yang, Chenjuan Guo, Yu Ma, and Christian S. Jensen. Toward personalized, context-aware routing. *The VLDB Journal*, 24(2):297–318, April 2015. ISSN 1066-8888. doi: 10.1007/s00778-015-0378-1. URL http://dx.doi.org/10.1007/s00778-015-0378-1.

[8] Stephan Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002. ISSN 1573-7624. doi: 10.1023/A:1020853410145. URL http://dx.doi.org/10.1023/A:1020853410145.

[9] Stephan Winter. *Route Specifications with a Linear Dual Graph*, pages 329–338. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. ISBN 978-3-642-56094-1. doi: 10.1007/978-3-642-56094-1_24. URL http://dx.doi.org/10.1007/978-3-642-56094-1_24.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. ISSN 00359246. URL http://www.jstor.org/stable/2984875.

[11] E. W. Dijkstra. A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK*, 1(1):269–271, 1959.

[12] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning in road networks. *Transportation Science*, 2015. URL http://dx.doi.org/10.1287/trsc.2014.0579.

[13] Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *J. Exp. Algorithmics*, 21(2):1.5:1–1.5:49, April 2016. ISSN 1084-6654. doi: 10.1145/2886843. URL http://doi.acm.org/10.1145/2886843.

[14] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks*, pages 319–333. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-68552-4. URL http://dx.doi.org/10.1007/978-3-540-68552-4_24.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313.

[17] Jon Froehlich and John Krumm. Route prediction from trip observations. In *SAE Technical Paper*. SAE International, 04 2008. doi: 10.4271/2008-01-0201. URL http://dx.doi.org/10.4271/2008-01-0201.

# Appendix

## A File System Configuration

The database tablespace for the dataset used in this paper (including raw GPS records, map matched trips, corresponding road maps, and table indexes) consumes approximately 1 TB of disk space. When querying such large datasets, the performance is often I/O bound. Therefore, disk read/write speed is essential for query speed. To increase fault tolerance as well as the read/write speed of our disks, we utilized both RAID 1 (mirror) and RAID 0 (stripe) with four 750 GB disks using a RAID 1+0 configuration. The RAID partitions are defined at the software level using ZFS (Zettabyte File System) which also supports file system compression.

An advantage of compression on the file system level, is that it allows transparent data compression for applications which would otherwise not support compression, e.g. PostgreSQL. The tradeoff between compression/decompression time and compression ratio depends on the compression method used. During this project, we favored fast compression/decompression time over a high compression ratio which is why we chose the LZ4 compression method[3]. LZ4 provides very fast lossless compression and even faster decompression with good scaling on multi-core CPUs. LZ4 provides a compression speed of 400 MB/s per core with decompression often limited by the RAM speed in multi-core systems.

### A.1 Compression Performance Experiment

We perform an experiment to measure the compression ratio and average query speed of four different query types: sequential scan, spatial index scan, index scan with nested loop, and bitmap index scan. We obtained a compression ratio of 3.92 with the run-time results of the experiment listed in Table 8. As the results shows, file system compression using LZ4 both improved disk space usage and query execution time (on average about 50 %), with the only disadvantage being decreased disk write speed.

| Query type | Compression | |
|---|---|---|
| | None | LZ4 |
| Sequential scan | 2339.8 | 915.4 |
| Sequential scan, with nested loop | 30.4 | 32.3 |
| Bitmap scan | 425.8 | 273.5 |
| Index scan, with nested loop | 279.2 | 284.2 |

Table 8: Average query execution time (in ms) over 3 runs of each query type both with and without LZ4 compression.

## B Machine Learning Techniques

### B.1 Learning Rate Decay

In machine learning, the learning rate, $\alpha$, refers to the amount of adjustment of weights and biases per iteration. A low learning rate results in a slower arrival at an optimum. With a high learning rate the learning time is decreased, but when set too high the algorithm can oscillate between the boundaries of an optimum as it continuously steps over the optimal point.

$$\alpha(i) = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min})e^{(-i/d)} \tag{5}$$

In equation 5 an exponentially decaying learning rate is defined given a lower and upper bound of the learning rate ($\alpha_{\min}$ and $\alpha_{\max}$), an iteration counter $i$, and a decay speed $d$. With a decaying learning rate, the algorithm will quickly descend towards the optimum, but as training progresses it will change less and less allowing for a finer tuning of the weights and biases.

### B.2 Dropout

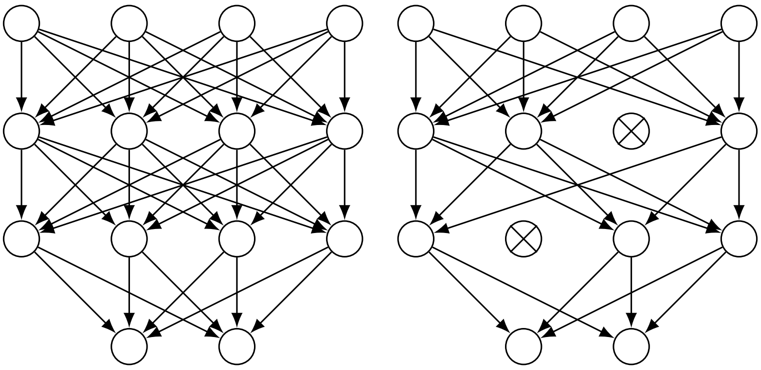
Overfitting is a complex problem in machine learning where the trained model is too complex to generalize the test set. Regularization functions try to prevent overfitting by generalizing parts of the trained model.

[3] github.com/lz4/lz4

Dropout is a relatively simple, but effective, regularization method for neural networks described by Srivastava et al. [16]. Dropout work by temporarily "freezing" a proportion of the neuron in a layer (hidden or input layer) for one training iteration, meaning their weights and biases are unchanged after the iteration and set to zero during the iteration.

A visualization can be seen in Figure 9, where Figure 9a shows a normal neural network before applying dropout. After applying dropout on both hidden layers, with a 25 % probability of freezing each neuron, the neural network could temporarily look like Figure 9b for a single training iteration.

Dropout has proven to be very useful in reducing overfitting in neural networks [16].



(a) Example of neural network. (b) After applying dropout.

Figure 9: Example of the effects of dropout with a 25 % probability of freezing each neuron.

# C Data Analysis

## C.1 Data Distribution

Figure 10 shows the trip distribution across monthly intervals between 2012-04 and 2014-12.

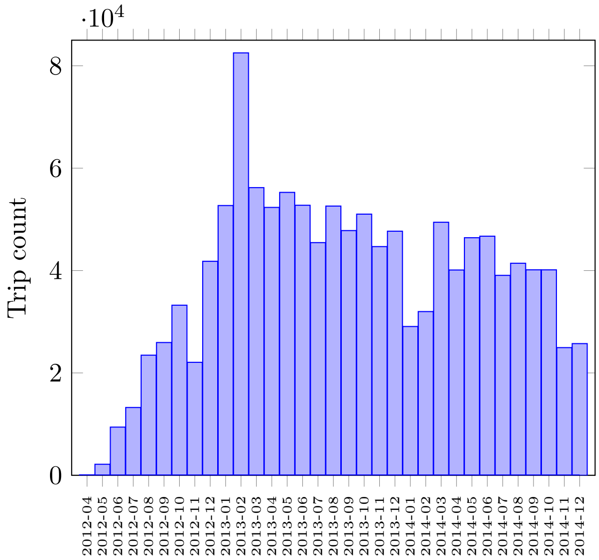


Figure 10: Density of trips across monthly intervals between 2012-04 and 2014-12.

Figure 11 illustrates the distribution of GPS observations in North Jutland for each road segment. Road segment with less than 25 observations are not shown.

## C.2 Data Filtering

As described in Section 5.1 we filter out short trips as these are unlikely to have much potential for personalization. Delling et al., who infers route preferences from American GPS traces in [2], filters out trips shorter than 483 meters and with a duration of less than 3 minutes. In lack of Danish threshold analysis, we base our threshold on the American threshold defined in [2]: trips must be at least 500 meters long and have a duration longer than

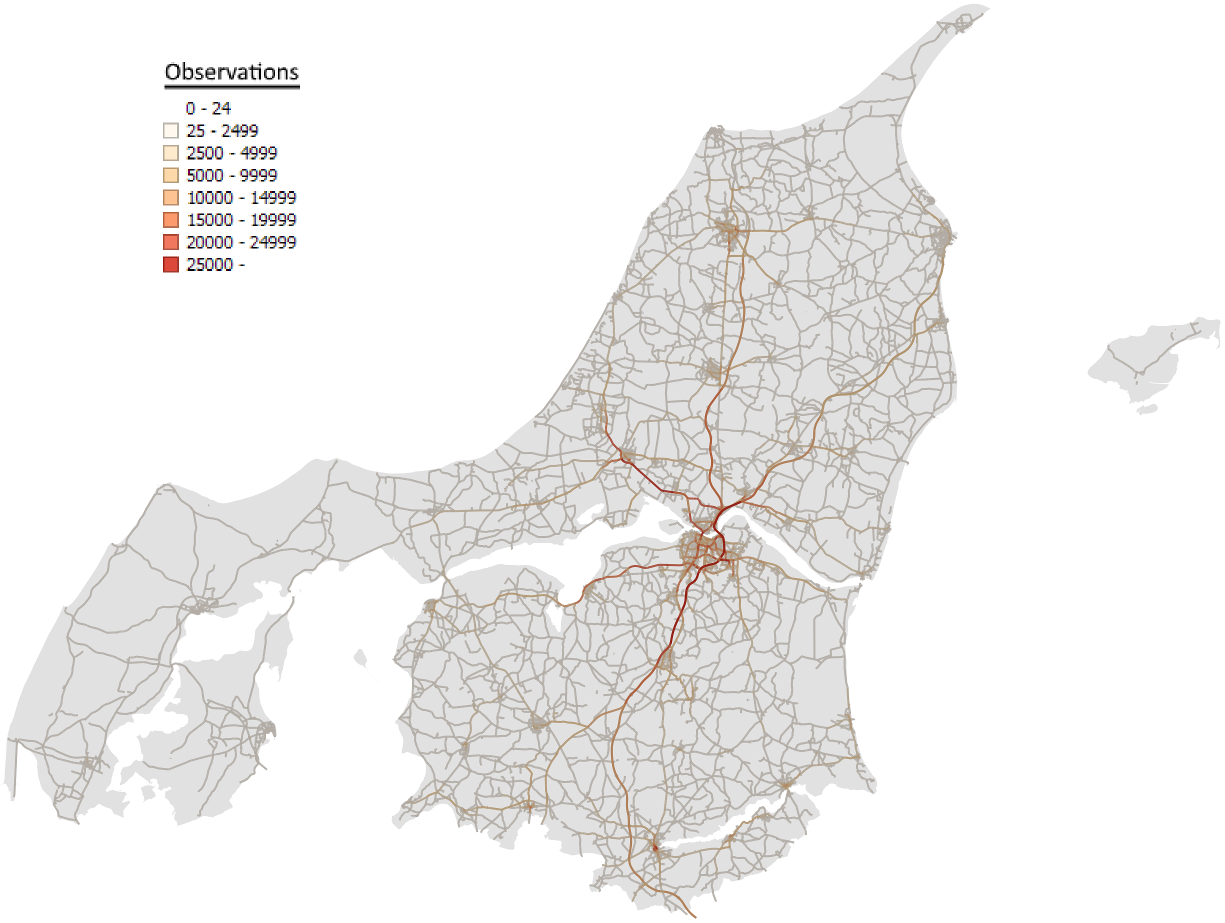


Figure 11: Usage of road segments in North Jutland with an observation count above 25.

3 minutes. In Figure 12a and Figure 12b the cumulative trip count is visualized for both minutes and kilometers driven per trip. The thresholds (3 minutes in Figure 12a and 500 meters in Figure 12b) are shown in red along with the percentage of trips removed using the threshold. After both filtration steps the total number of trips is reduced from 1267 K to 958 K (a reduction of 25.1 %).

## C.3 Coupling Data Based On Different Map Versions

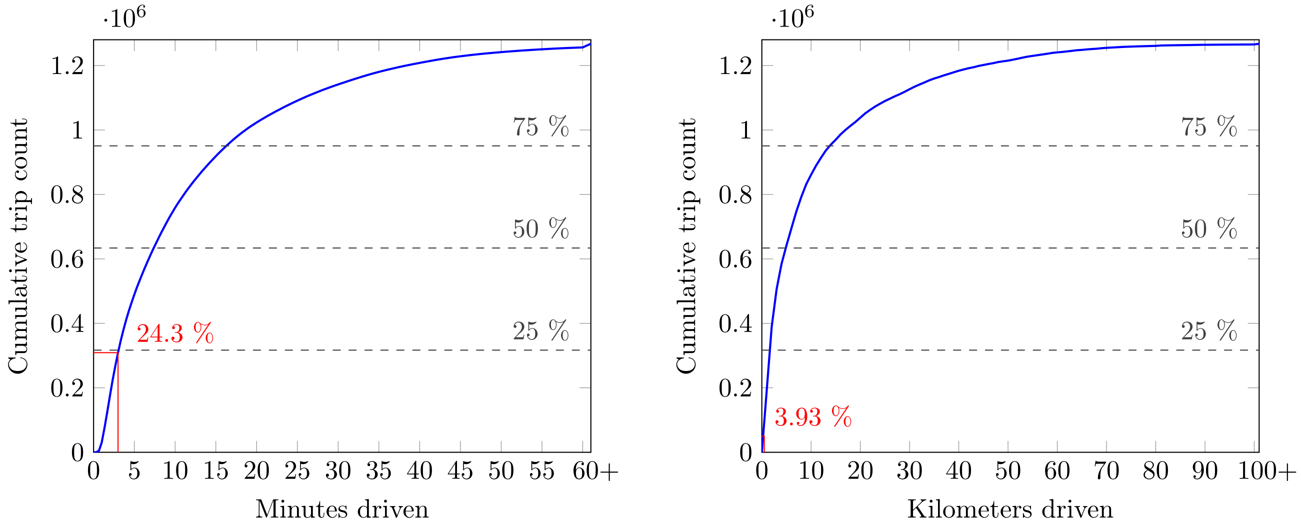
As described in Section 5.1 we use data from the ITS project, that is map matched to the 2014-01-01 version of OSM. We also use a speedmap provided by Aalborg University, based on the 2015-10-06 version of OSM. A difference in time of 22 months between the two maps makes coupling the datasets difficult, as OSM changes rapidly.

To handle this problem, we invented a 4-step mapping solution from OSM 2015 to OSM 2014 which checks the following conditions in order:

1. *ID match*: If the road segments have the exact same ID, the segments are assumed to be either identical or sub segments of one another.
2. *Spatial join*: If the the road segment from OSM 2014, with an extra buffer of 50 meters, spatially contains road segments from OSM 2015 we map to the 2015 road segment with the smallest distance[4].
3. *Reverse spatial join*: Follows the same principle of the previous step but from OSM 2015 to OSM 2014.
4. *Neighbor inference*: If a segment in OSM 2014 have neighboring segments with a mapping, assume same speed as neighbouring segment.

After removing road segments from outside North Jutland (as described in Section 5.1) we start the mapping with 111 K non-mapped road segments, with the result of each step outlined in Table 9. After applying the

[4] Distances are calculated using the route similarity method presented by Froehlich and Krumm in [17].



(a) The cumulative distribution of trips ordered by their duration in minutes, with a suitable cutoff value of 3 minutes.

(b) The cumulative distribution of trips ordered by their length in kilometers, with a suitable cutoff value of 1 kilometers.

Figure 12: Filtering of trips.

4-step mapping solution, 99.95 % of all OSM 2014 road segments in North Jutland have been mapped to an appropriate OSM 2015 road segment. Out of the remaining 53 road segments, we were able to manually map 11 of them. The remaining segments were either completely missing from the OSM 2015 or 2014 map.

| Mapping method | Mapped | Non-mapped | Mapped percentage |
|---|---:|---:|---:|
| Before mapping | 0 | 111,098 | 0.00 % |
| (1) ID matched | 108,976 | 2,235 | 97.99 % |
| (2) Spatial join | 110,737 | 474 | 99.57 % |
| (3) Reverse spatial join | 110992 | 219 | 99.80 % |
| (4) Neighbor inference | 111045 | 53 | 99.95 % |

Table 9: Mapping results throughout the 4 steps.

# D Cluster Preference Matrix

| Cluster | Trips | Duration | Distance | Bridges | Tunnels | Obstacles | Traffic sig. | Left turns | Right turns | U-turns | Motorways | Rural roads | Other roads |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 15.89 % | 1.654e−4 | 3.434 | 2.053e−4 | 3.404e−4 | 5.325e−5 | 1.112e−5 | 1.114e−4 | 1.947e−4 | 1.169e−6 | 1.211e−1 | 7.891e−1 | 3.319 |
| 2 | 16.56 % | 7.636e−5 | 2.727 | 7.732e−5 | 1.399e−4 | 6.366e−5 | 9.928e−5 | 2.897e−1 | 9.520e−5 | 6.005 | 1.556e−5 | 3.639e−1 | 2.551 |
| 3 | 7.16 % | 8.261e−5 | 2.673 | 1.339e−4 | 6.467e−5 | 8.792e−5 | 1.331e−4 | 1.116e−4 | 1.203e−4 | 1.500e−4 | 1.089e−5 | 2.466e−1 | 4.040 |
| 4 | 6.18 % | 2.421 | 2.272 | 5.766e−5 | 1.854e−2 | 2.108e−2 | 2.036e−3 | 7.427e−1 | 7.716e−5 | 1.819e−1 | 3.492e−2 | 2.473e−1 | 2.149 |
| 5 | 6.23 % | 1.906e−5 | 1.631 | 7.334e−1 | 9.995e−2 | 5.578e−5 | 1.838e−4 | 4.960e−5 | 6.938e−5 | 6.894e−5 | 1.240 | 6.547e−2 | 1.645 |
| 6 | 4.17 % | 1.574e−4 | 1.656 | 2.436e−4 | 3.186e−5 | 5.162e−5 | 3.555e−5 | 3.075 | 3.047 | 4.260e−5 | 3.975e−2 | 3.678e−1 | 1.692 |
| 7 | 3.25 % | 5.301e−4 | 4.403e−5 | 2.347e−3 | 1.217e−4 | 5.841e−4 | 1.680e−3 | 1.962e−4 | 8.797e−4 | 2.492e−4 | 1.994e−4 | 3.411 | 9.119e−4 |
| 8 | 5.54 % | 4.688 | 3.085e−5 | 5.403e−5 | 2.007e−4 | 8.426e−5 | 2.399e−4 | 3.278e−5 | 9.951e−5 | 5.173e−5 | 1.484e−2 | 6.297e−2 | 6.847e−5 |
| 9 | 4.38 % | 2.220e−4 | 1.906e−1 | 2.429e−2 | 2.207e−4 | 2.372e−4 | 2.479e−4 | 1.557 | 7.516e−1 | 7.452 | 1.421e−2 | 1.625e−1 | 6.671e−1 |
| 10 | 1.94 % | 1.307e−4 | 5.452e−4 | 2.349e−5 | 1.535e−4 | 3.508e−4 | 1.394e−1 | 3.588 | 3.588 | 4.060e−2 | 7.951e−3 | 1.185e−4 | 2.584e−4 |
| 11 | 3.41 % | 9.766e−5 | 1.762 | 5.019e−5 | 1.949e−4 | 9.896e−5 | 3.853e−2 | 2.852 | 8.553e−5 | 3.858 | 4.093e−2 | 3.757e−1 | 2.170 |
| 12 | 2.95 % | 5.093e−4 | 3.132 | 1.354e−4 | 1.464e−4 | 6.929e−5 | 1.472e−1 | 6.787e−5 | 5.119e−6 | 5.581e−5 | 6.170e−2 | 5.327e−1 | 2.828 |
| 13 | 8.30 % | 1.983e−4 | 3.077e−4 | 1.763e−4 | 3.034e−5 | 1.050e−4 | 1.630e−4 | 6.115e−4 | 2.081e−4 | 2.619e−3 | 6.573e−5 | 4.410e−4 | 7.704 |
| 14 | 6.47 % | 5.616e−3 | 1.969e−4 | 8.844e−5 | 4.239e−5 | 1.050e−4 | 7.606e−4 | 8.351e−3 | 7.588e−3 | 9.300 | 2.749e−4 | 5.779e−4 | 1.556e−3 |
| 15 | 2.60 % | 5.675e−5 | 1.228e−2 | 7.877e−5 | 3.235e−5 | 2.752 | 1.266e−4 | 1.116e−4 | 1.384e−3 | 4.768e−1 | 6.775e−3 | 5.968e−2 | 3.315e−1 |
| 16 | 4.97 % | 1.287 | 1.316e−4 | 3.091e−5 | 5.653e−5 | 4.875e−5 | 5.181e−5 | 6.779e−2 | 4.813e−5 | 8.020 | 2.475e−3 | 1.713e−4 | 3.987e−1 |

Table 10: Preferences towards the 12 road features for the 16 identified clusters. High values (weights) indicate higher cost of traversing router with the given feature.