Master Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Automated refinement of bicycle routing graphs using ride tracking data

Bc. Jan Mayer

Supervisor: Doc. Ing. Michal Jakob, Ph.D.
May 2022

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Mayer Jan**	Personal ID number:	**474529**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence**		

II. Master’s thesis details

Master’s thesis title in English:

Automated refinement of bicycle routing graphs using ride tracking data

Master’s thesis title in Czech:

Automatické zpřesňování cyklistických navigačních grafů ze záznamů dat o projetých trasách

Guidelines:

Maps used as the basis of bicycle navigation systems do not always accurately represent the attributes of the real-world bicycle route network. Such deficiencies of input map data can be compensated by analyzing data recorded from mobile phone sensors of bicycle riders, such as geolocated tracks or accelerometer data. The goal of the thesis is to explore the possibility to use such rider-generated data to improve the quality of bicycle routing graphs.
1) Survey existing approaches to improving navigation map data from recorded ride data.
2) Formally define the problem of automated routing graph refinement from recorded ride data.
3) Propose a suitable algorithm for solving the defined problem.
4) Implement the proposed algorithm.
5) Evaluate the algorithm on the provided samples of recorded ride data.

Bibliography / sources:

[1] Heidt, Johannes, and Klaus Dorer. "Classification and Prediction of Bicycle-Road-Quality using IMU Data." In Artificial Intelligence: Application in Life Sciences and Beyond, pp. 138-149. 2021.
[2] Bigazzi, Alexander, and Robin Lindsey. "A utility-based bicycle speed choice model with time and energy factors." Transportation 46, no. 3 (2019): 995-1009.
[3] Hoffmann, Marius, Michael Mock, and Michael May. "Road-quality Classification and Bump Detection with Bicycle-Mounted Smartphones." In UDM@ IJCAI, p. 39. 2013.
[4] Litzenberger, Stefan, Troels Christensen, Otto Hofstätter, and Anton Sabo. "Prediction of road surface quality during cycling using smartphone accelerometer data." In Proceedings of The 12th Conference of the International Sports Engineering Association, vol. 2, no. 6, p. 217. 2018.

Name and workplace of master’s thesis supervisor:

doc. Ing. Michal Jakob, Ph.D.   Artificial Intelligence Center FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **28.01.2022**   Deadline for master's thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

doc. Ing. Michal Jakob, Ph.D.	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature		Dean’s signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to express my gratitude to my supervisors Michal Jakob and Pavol Žilecký for providing guidance and feedback throughout this project. I would also like to thank my family for supporting me during my work on this thesis and during my entire studies at CTU.

Declaration

Abstract

Surface quality is one of the most important road attributes in the context of path planning for bicycle users. Unfortunately, this attribute is often incorrectly filled in or missing from the used map data. Many published approaches successfully used data gathered using high-frequency sensors such as gyroscopes or accelerometers to model the surface quality. However, gathering these measurements at large scale is problematic, and the resulting datasets are impractical to work with due to their size. In this thesis, we used data in the form of GPS measurements gathered by cycling volunteers. Our model based on graph neural networks successfully predicted the surface quality achieving RMSE of 0.72. We used the model to predict the surface quality of the roads and paths with inaccurate information and manually validated the results in several places. In this thesis, we showed that using GPS measurements is a viable option to model the surface quality in large areas.

Keywords: navigation graph, GPS measurements, cycling, machine learning, graph neural networks

Supervisor: Doc. Ing. Michal Jakob, Ph.D.
FEE, Department of Computer Science

Abstrakt

V kontextu plánování cyklistických tras je kvalita povrchu jednou z nejdůležitějších vlastností cest. Tento atribut je bohužel často v používaných mapách špatně vyplněn nebo úplně chybí. V minulosti byla kvalita povrchu úspěšně modelována pomocí dat nasbíraných z vysokofrekvenčních senzorů jako je gyroskop nebo akcelerometr. Nasbírat tato data ve velkém měřítku je však problematické a práce se získanými daty je kvůli jejich velikosti komplikovaná. V této práci jsme použili data ve formě GPS měření nasbíraných dobrovolníky. Náš model založený na grafových neuronových sítích úspěšně modeloval kvalitu povrchu a dosáhl chyby RMSE v hodnotě 0.72. Tento model jsme použili k predikci kvality povrchu na cestách s nepřesnými informacemi a manuálně zkontrolovali několik výsledků. V této práci jsme ukázali, že GPS měření jsou použitelná k predikci kvality povrchu na velkém území.

Klíčová slova: navigační graf, GPS měření, cyklistika, strojové učení, grafové neuronové sítě

Překlad názvu: Automatické zpřesňování cyklistických navigačních grafů ze záznamů dat o projetých trasách

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Problem and motivation

Riding a bicycle has become very popular as a way to exercise and relax as well as a viable way to travel in cities with increasingly heavier traffic. There are many reasons to use bicycles over cars such as to reduce the pollution[38] and noise[10], both of which have negative effects on health. Another reason may be the benefits of regular exercise or convenience. Some European cities are famous for successfully making the city easy to travel by a bicycle [47].

Since the boom of smartphones, many different apps enhancing the experience and the effectiveness of riding a bicycle have become popular. A very important role is played by the navigation apps. These apps, just like their automotive counterparts, aim to provide the user with a way to plan paths based on the user's preferences. The main priority of car drivers is usually the time of travel. For bicycle riders, there are many more desired properties of the path, such as elevation, traffic, or surface quality. These properties depend on the cyclist's preferences, physical capabilities, or bike type, which significantly vary across different cyclists. Moreover, these properties have the potential also to affect the travel time more dramatically. It is thus crucial for the apps to work with the correct information about the real-world state of the roads. Unfortunately, the maps currently used often contain incomplete or wrong information.

In this thesis, we will focus on the attribute of the edges, which rates the quality of the surface, which is a very important attribute in the context of planning a path for a bicycle, but an attribute that is very often missing or incorrectly filled in the map data. Many successful approaches have been proposed in the past, which used the data from high-frequency sensors to predict the road surface. However, this kind of data is difficult to collect and work with at a large scale. In this thesis, we will try to model the surface quality using the data from the GPS measurements, which consist of the position and the velocity of the user and can be gathered using a wide variety of devices such as smartphones, smartwatches, or fitness trackers.

The goal of the thesis is to propose, implement and evaluate an algorithm that will be able to automatically improve the accuracy of the surface quality label in the given map using the user-generated tracking data.

1.2 Thesis structure

This thesis consists of eight chapters:

1. **Introduction** – in this chapter, we introduce the problem of the automated graph refinement and the motivation for its solution.
2. **Related work** – in this chapter, we summarize the published approaches to solving the problem.
3. **Task specification** – in this chapter, we formally define the problem and set the goal of this thesis.
4. **Theoretical background** – in this chapter, we describe the machine learning techniques which we used in the practical part of this thesis.
5. **Data and preprocessing** – in this chapter we describe our data sources and the process that led to creating the final dataset.
6. **Solution approach** – in this chapter, we describe the approach we have chosen to solve the problem and some technical details regarding the used techniques.
7. **Results** – in this chapter, we evaluate our approach on the available data.
8. **Conclusion** – in this chapter, we summarize the results of this thesis and discuss the limitations of the selected approach.

Chapter 2
Related work

We can divide the existing approaches into two main categories. The first category aims to improve the topology of the navigation graph. The second category focuses on correcting or adding node and edge attributes.

2.1 Improving the graph topology

Due to the constant change of the road networks caused by the construction of the new roads or demolition of the existing ones, the navigation graph has to be periodically updated. However, doing so manually would require large amounts of human work. For this reason, great efforts have been made to update the navigation graph automatically.

The authors of [25] used GPS trajectories gathered by car navigation devices to update the existing digital maps. They presented a method to detect new roads and roundabouts. In [26] the method was extended to detect incongruences of different road geometry, alterations of junction positions, prohibited maneuvers, and traffic directions. Authors of [19] used an artificial neural network model to add new roads to the network, based on the GPS data collected from the car-navigation systems of the users. In [5], a system capable of creating the navigation graph entirely from the GPS measurements was developed. The system was proven to be working correctly by showing the generated and real graph to be identical.

It is also possible to use satellite imagery to update the the navigation graph. A framework for road change detection and map updating based on imagery was proposed in [59] and [6].

2.2 Adding and correcting attributes

The navigation graph can contain attributes useful for using the map, such as the types of the roads or the position of the points of interests. The approaches to the modification or creation of the graph attributes can be divided into three categories based on the type of data that is used.

2.2.1 Attribute correction based on GPS data

The GPS measurements provide precise information about the travel of the user in the road network by measuring the geographical coordinates and the velocity of the user. To collect such data, either smartphones or car navigations are used.

In [15] the authors presented a simple method based on the data analysis and thresholding, which is able to estimate the location of stop signs and location/timing of traffic lights with more than 90% accuracy from the GPS data.

The GPS data can also be useful to model the traffic condition, which is a very important feature for path planning. In [56], the authors propose a simple yet effective way to model the traffic condition based on measuring statistics such as the average velocity or the stopping times.

2.2.2 Attribute correction based on images

Another type of data that can be used is satellite or aerial imagery. Its biggest advantage is that, unlike the GPS measurements, the satellite imagery already covers the majority of the Earth's surface, so there is no need for data collection. The disadvantage is that to infer some information from the pictures, more sophisticated models and computational power is usually needed. Moreover, the resolution of the satellite and aerial imagery may not be high enough to reliably recognize the different types and qualities of the surface.

In [12] the satellite imagery is used to predict the road quality using a convolutional neural network (CNN), which achieves the accuracy of 73%. The authors of [29] apply a similar approach but also use a graph neural network model to predict the attributes using the embeddings created by a CNN from the satellite images.

2.2.3 Attribute correction based on data from accelerometer and/or gyroscope

The type of data which provides perhaps the most accurate information about the nature of the travel can be obtained from sensors such as gyroscope and accelerometer. This type of data is especially useful for predicting the type or quality of the surface of the road. The disadvantage of this type of data is that of the three presented types of data, it is perhaps the hardest one to collect at a large scale and the most sensitive to the equipment used to collect it, For example, different bicycle forks or tires can transmit the vibrations of the road very differently, as was studied in [32].

In [30] a special device mounted on a bicycle handlebar containing an accurate accelerometer and gyroscope was used. In [35] a smartphone built-in accelerometer was used. Both approaches achieved a very high accuracy of 99% and 97%, respectively. However, the model from [30] was unable[1] to

[1]It achieved only 26% accuracy.

correctly classify the data from a different type of bicycle, and in [35], the authors did not evaluate the model's transferability to a different bicycle. This indicates that it may be problematic to predict the road surface using data generated by users riding different types of a bike without some sophisticated data preprocessing. The sensor does not need to be mounted on a bicycle. The authors of [22] used the data from the vibration sensors in the smartphone placed in the car to compute the IRI (international roughness index) [43].

2.3 Difference in approach in this thesis and the previous work

The main difference between our approach and the previous work is in the used data. Our user-generated dataset does not contain measurements from sensors other than GPS. In the previous subsections, we cited successful attempts to model the surface type or quality using data from an accelerometer or gyroscope. However, these measurements are hard to collect at such a large scale. Another problem is the amount of data these high-frequency sensors produce. While the GPS measurements in our data were taken roughly every two seconds, the number of measurements these sensors make can be up to hundreds per second. We believe that the relationship between the surface quality and travel velocity, together with other road properties, is strong enough to reliably model the quality of the surface.

Chapter 3

Task specification

The task of the graph refinement consists of taking a navigation graph and modifying it in such a way that it describes the real-world transport network more accurately. In this chapter, we will first formally define this concept. Then we will present an overview of the existing approaches. Finally, we will state the goal of this thesis.

3.1 Formal problem definition

In this section we will formally define the general task of the graph refinement.

3.1.1 Definitions

Navigation graph

The *navigation graph* G is a tuple (V, E, f_V, f_E), where

- V is a finite non-empty set of vertices. Also denoted by $V(G)$.
- E is a finite set of edges. Also denoted by $E(G)$
- (V, E) form a simple directed graph, which means
  1. The edges are ordered pairs of vertices, where $e = (v_1, v_2) \in E$ denotes that edge e starts at v_1 and ends at v_2.
  2. There are no multiple edges. This holds automatically as we defined E to be a set.
  3. There are no loops - $\forall (v_1, v_2) \in E : v_1 \neq v_2$.
- $f_V : V \to \mathbb{R}^k$ is a function assigning a feature vector to each vertex. Typically the features are latitude, longitude and elevation.
- $f_E : E \to \mathbb{R}^l$ is a function assigning a feature vector to each edge. The features are typically some real-world properties of the edge, such as type of surface or maximal allowed speed.

Navigation graphs are used to represent real-world *transport network*s. In the navigation graphs, we can use the graph-search algorithms to find paths with desired qualities (such as distance, time travel or traffic density).

Trip

The *trip* $t = (x, M)$ is a pair of a vector x and an ordered list $M = (m_1, m_2, \dots, m_n)$ of measurements $m_i = (\text{lat}, \text{lon}, v, ts, b)$, where

- x contains meta information about the whole trip, such as user id or type of the bike, that was used.
- lat is the measured latitude.
- lon is the measured longitude.
- v is the measured velocity.
- ts is the measured timestamp. Timestamps within a trip are non-decreasing with respect to the order of the measurements.
- b is the measured bearing in degrees.

The trip t represents real-world movement of a user from start $(\text{lat}_1, \text{lon}_1)$ to destination $(\text{lat}_n, \text{lon}_n)$.

3.1.2 Task

Input

The instance of the problem I is a tuple $I = (G, T, d)$, where

- $G = (V, E, f_V, f_E)$ is a navigation graph.
- $T = \{t_1, t_2, \dots, t_n\}$ is a set of trips.
- $d : \mathcal{G} \times \mathcal{G} \to \mathbb{R}_0^+$ is a *distance function* on $\mathcal{G}$, which is the space of all navigation graphs. d satisfies the following axioms for all $G_1, G_2, G_3 \in \mathcal{G}$:

  1. $d(G_1, G_2) = 0 \iff G_1 = G_2$
  2. $d(G_1, G_2) = d(G_2, G_1)$
  3. $d(G_1, G_2) + d(G_2, G_3) \geq d(G_1, G_3)$

Admissible solution

An admissible solution to the problem instance I is an updated navigation graph $G' = (V', E', f'_V, f'_E)$, where

- $V' = V \cup V_{new}$ and V_{new} contains newly added vertices.
- $E' = E \cup E_{new}$ and E_{new} contains newly added edges.
- $f'_V : V' \to \mathbb{R}^k$ assigns new features to the vertices from V_{new} and preserves the features of the vertices from V $(\forall v \in V : (f_v(v) = f'_v(v))$.
- $f'_E : E' \to \mathbb{R}^{l'}$ assigns new features to the edges from E_{new} and also possibly corrected features to the edges from E. It can also introduce some new features.

Goal

Let G^* be a navigation graph that describes the real-world properties of the transport network completely, without any mistakes or simplifications. Given the problem instance $I = (G, T, d)$, the goal is to find an admissible solution G', such that $d(G', G^*) < d(G, G^*)$. Ideally, we would like the value of $d(G', G^*)$ to be as close to zero as possible. Note, that $d(G', G^*) = 0 \iff G' = G^*$.

In other words, we want to find a G', which describes the real-world properties of the transport network better than G. To do that, we will use the user-generated trips T.

3.2 Goal of the thesis

In this thesis, we will focus on correcting the edge features. More specifically, we will only focus on one of the features, namely, the surface quality. V_{new} and E_{new} will then be empty sets, and the distance function d will only compare the differences between the edges surface quality. Let $G' = (V, E, f_V, f'_E)$ and $G^* = (V, E, f_V, f^*_E)$ be two navigation graphs which differ only in edges' surface quality denoted as $f_E(e)_S$. More formally, for the functions f'_E and f^*_E, the following holds:

$$\forall(e \in E, i \neq S) : (f^*_E(e)_i = f'_E(e)_i).$$

The distance between the two graphs will be defined as

$$d(G', G^*) = \frac{1}{|E|} \sum_{e \in E} (f'_E(e)_S - f^*_E(e)_S)^2,$$

i.e., the mean squared difference between the edges' surface qualities. It is easy to see that d satisfies the axioms from the input specification 3.1.2.

We will approach this problem as a machine learning problem. We will divide the edges of the input graph into two categories: edges with the correct label and edges with an incorrect or missing label. We will train a model of the surface quality using the former part of the edges and then use the model to correct or fill in the labels of the latter part of the edges.

Even though by only focusing on one edge feature, we significantly restrict the solution space of the general problem we defined in the previous section, the surface quality is one of the most important properties of the roads and paths in path-planning, especially for bike riders. Improving the accuracy of this feature in the navigation graph will lead to a better experience for the bike users planning their trips in the graph.

Chapter 4

Theoretical background

In this chapter, we will describe the theory behind the tasks and models, which we will then use to solve our problem. The chapter aims to provide a thorough explanation of the used terms so that the reader can understand the inner workings of the used machine learning algorithms as well as the choices made during the design of the algorithm solving our task. The first section of the chapter formally defines the machine learning tasks, and the second section describes three models which can be used to solve the defined tasks. In the final section, we describe the evaluation metrics that will be used in the practical part of the thesis.

4.1 Machine learning tasks

This section formally defines three machine learning tasks – the general prediction task, the classification task as a case of the prediction task, and finally, the task of ordinal regression as a modification of the classification task.

4.1.1 Prediction task

We present the formal definition of the prediction task as it is defined in [23]. Let

- $\mathcal{X}$ be a set of input observation,
- $\mathcal{Y}$ be a set of hidden states,
- $h : \mathcal{X} \to \mathcal{Y}$ be a prediction strategy,
- $(x, y) \in \mathcal{X} \times \mathcal{Y}$ be samples independently[1] drawn with p.d.f. $p(x, y)$,
- $\ell : \mathcal{X} \times \mathcal{Y}$ be a loss function.

[1]Notice that this assumption is violated in our data – the connected roads have higher probability of having similar properties.

Find a strategy with the minimal *expected risk*

$$R(h) = \int_{(x,y)\in(\mathcal{X}\times\mathcal{Y})} \ell(y, h(x))p(x,y)\; dx.$$

In some cases, when $p(x,y)$ is known, we can solve the problem exactly and without any samples drawn from $\mathcal{X}\times\mathcal{Y}$. However, in most cases, $p(x,y)$ is unknown. In this case, we work with a *training set* of samples

$$\mathcal{T} = \{(x_i, y_i) \in (\mathcal{X}\times\mathcal{Y}) \mid i = 1, \ldots, m\}$$

drawn independently from the same distribution $p(x,y)$. Then using the training set we find a strategy h. Even though $R(h)$ cannot be computed, it can be estimated by the *empirical risk* on a *test set* $\mathcal{S}$

$$R_{\mathcal{S}}(h) = \frac{1}{|S|}\sum_{(x,y)\in\mathcal{S}} \ell(y, h(x)).$$

In practice, h is usually found by minimizing the empirical risk on the training set $\mathcal{T}$ with loss function ℓ', which may but may not be the same as ℓ.

Because the training set contains labeled data, we talk about *supervised learning*.

4.1.2 Classification

The classification task is a prediction task where the goal is to classify the observations into two (binary classification) or more (multi-class classification) classes. This means that without the loss of generality, we can assume that $\mathcal{Y} = \{1, \ldots, k\}$, where k is the number of classes. A loss function of a classification problem is a function that penalizes the model for assigning a sample to an incorrect class. An example of such a function would be the 0-1 loss:

$$\ell(y, h(x)) = \begin{cases} 0 & y = h(x) \\ 1 & \text{otherwise} \end{cases}$$

Some models do not predict the class directly, but rather output a probability distribution over $\mathcal{Y}$: $q = (q_i, \ldots, q_k)$. When the desired output is also a distribution over $\mathcal{Y}$: $p = (p_i, \ldots, p_k)$, the cross entropy loss function can be used:

$$\ell(p,q) = -\sum_{i=1}^{k} p_i \log(q_i).$$

Notice that if the task is to assign each sample to exactly one class as we defined it above, the distribution for a given sample with the correct class c looks like this: $p = (\underbrace{0, \ldots, 0, 1}_{c}, 0, \ldots, 0)$. The loss can be simplified as

$$\ell(p,q) = -\log(q_c).$$

We can see that the functions penalize the model for not predicting the right class. If we look at both loss functions, we can see that they do not take into account which class was predicted instead of the right one. In many cases, this is the desired behavior. However, there are cases where we are able to define ordering between the classes and would like the model to predict the label as close as possible to the right one. This is the task of ordinal regression.

4.1.3 Ordinal regression

Ordinal regression (or ordinal classification) is a task similar to classification. The main difference is that it takes into account the given ordering of the classes. An example of such an input would be to classify the student's exam into grades A-F. In this case, when the correct output is A, the model should be penalized more for predicting E than for predicting B. For this reason, ordinal regression has the properties of both classification and regression in the sense that we want the model to classify the samples to the closest class (ideally the correct one). There are two naive ways to solve this task:

- Ignore the ordering and use a classification model. If we are able to create a model, which very often classifies the samples correctly, we can get good results even though the model does not take the ordering of the classes into account.
- Use a regression model. We could make a model, which predicts a real number from $[1, k]$ and train it using a regression loss function, for example, mean square error. The predictions would be made by rounding the model's output to the nearest integer. The problem is in mapping the classes to the integers from 1 to k. When we do that, we assume that all classes are the same distance apart. Imagine we try to predict a test grade A-E. In this case, it may not make sense to assume that B is twice as bad as A.

Because neither of these approaches is ideal, a number of machine learning methods have been developed or redesigned to address ordinal regression, such as perceptron or support vector machines. [17]

One possible approach was introduced in [24]. Using this approach, we can solve the task of ordinal regression by solving $k-1$ binary classification tasks. The main idea of the approach is to encode the class labels as $k-1$ dimensional vectors. When the correct label of a sample is c, we encode it as $(t_1, t_2, \ldots, t_{k-1})$, where

$$t_i = \begin{cases} 1 & i < c \\ 0 & \text{otherwise} \end{cases}$$

The interpretation of this encoding is, that t_i is a probability, that the sample x belongs in a class $c' > i$, i,e, $t_i = Pr(c' > i|x)$. The probability of sample x belonging to a particular class c can be computed as

$$Pr(c) = \begin{cases} 1 - Pr(y > 1|x) & c = 1 \\ Pr(y > c-1|x) - Pr(y > c|x) & 1 < c < k \\ Pr(y > c-1|x) & c = k. \end{cases}$$

After encoding the labels as vectors, we train a binary classifier on each element of the vector. The final prediction can be made by computing the probability of each label and taking the most probable one.

In [17] instead of training $k-1$ separate classifiers a neural network with $k-1$ outputs is used instead. To guarantee that each output is in the interval $(0,1)$ the sigmoid function is applied to each output:

$$S(z)_i = \frac{1}{1+e^{-z_i}}$$

Now to train the neural network using backpropagation (subsection 4.2.2), we can use either the mean square error loss

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{k-1}\sum_{i=1}^{k-1}(y_i - \hat{y}_i)^2$$

or binary cross entropy loss[2] between the network's output and the encoded labels. From the model's output $o = (o_1, o_2, \ldots, o_{k-1})$ the predicted class can be obtained by computing $|\{i \mid \forall(j \leq i) : o_j > 0.5\}| + 1$, i.e. the number of outputs o_i larger than 0.5 from the left plus one. For example $(0.7, 0.8, 0)$ gets mapped to class 3 and $(0.1, 0.0, 0.0)$ gets mapped to 1.

One problem with this approach is that the model can output inconsistent predictions. Inconsistent prediction is a prediction whose interpretation contradicts itself. For example, output vector $(0.7, 0.0, 0.6)$ tells us that the sample belongs to a class higher than 2 with zero probability, but at the same time, it says that the sample belongs to a class higher than 3 with the probability of 0.6. An example of inconsistent prediction can be seen in Figure 4.1. This problem is addressed in [14], where a CORAL (COnsistent RAnk Logits) method is proposed. First, a custom output layer is proposed. The layer is similar to a standard linear layer, but each output has a separate bias. Moreover, a custom loss function is introduced. The authors provide theoretical guarantees that a network optimizing the proposed loss function will make consistent predictions.

4.2 Machine learning models

In this subsection, we will describe three types of models that can be used to solve the tasks defined in the previous section. The first one is the gradient boosting method. It assumes independent samples, but it is a very powerful tool for solving classification problems on tabular data [11]. The second type is the class of artificial neural networks (ANNs). ANNs are models whose design is inspired by the human brain. They can be used to solve almost any machine learning task. The third type of model is a class of graph neural networks (GNNs). A GNN is a special type of artificial neural network whose input consists of not only the feature vectors but also the underlying graph structure – the relationships among the samples.

[2]special case of the cross entropy loss (4.1.2) with only two labels

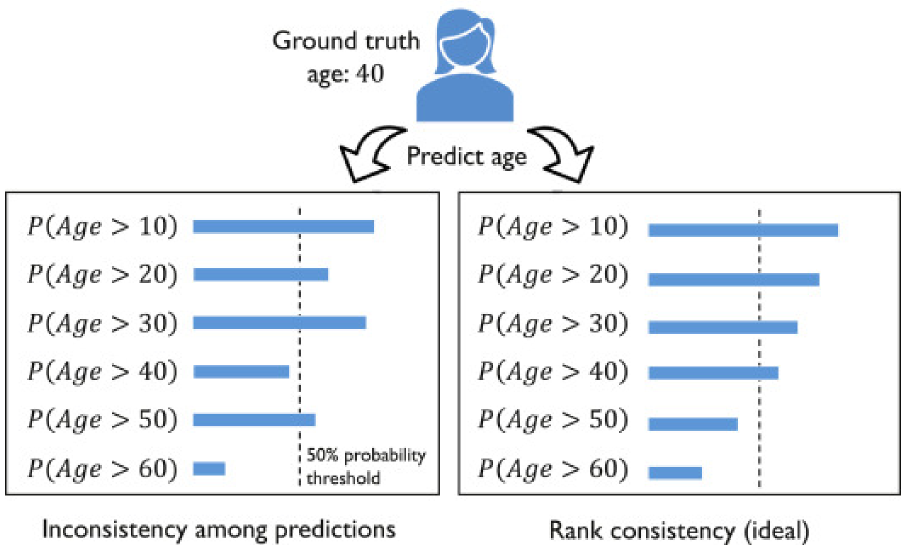


Figure 4.1: An example of inconsistent and consistent predictions for predicting an age group. Image taken from [14]

4.2.1 Gradient boosting method

Gradient boosting is an ensemble machine learning method that consists of iteratively training an ensemble of weak learners. The general algorithm, together with its application to optimizing different loss functions from both regression and classification tasks, was introduced in [27]. Algorithm 1 shows the procedure. The initial model is a constant model predicting the value minimizing the average loss on the training dataset. Then in each iteration, the gradient vector g_k is computed. It contains the partial derivatives of the loss function with respect to the predicted values. After that, a new weak learner is trained to predict the values of $-g_k$ with the squared error loss. Then the weak learner will be multiplied by coefficient β_k and added to the model from the ensemble model from the last iteration[3], forming the ensemble model for the current iteration. The coefficient β_k can either be found by line search or set constant for all iterations.

We can see that this general algorithm is able to optimize any differentiable loss function. For example, if we are solving a regression problem using least squares, we get the gradient vector

$$(g_k)_i = \frac{\partial}{\partial f_k(x_i)} \left(\frac{1}{2} (y_i - f_k(x_i))^2 \right) = f_k(x_i) - y_i.$$

Solving a classification problem with K classes is slightly more complicated. The loss function will be the cross-entropy loss described in 4.1.2. We will proceed by training K models. Let f_k be the kth model. Now we will apply

[3]by performing arithmetic on the models, we mean performing arithmetic on the corresponding predicted values

the softmax function to the outputs. The predicted probability of a sample x belonging to a class k will be computed as

$$q_k(x) = \frac{e^{f_k(x)}}{\sum_{i=1}^{K} e^{f_i(x)}}.$$

Substituting q_k into the loss function, we get the gradient for the kth model:

$$(g_k)_{i,k} = \frac{\partial}{\partial f_k(x_i)} \left[-\sum_{c=1}^{K} y_{i,c} \log(q_c(x_i)) \right] = p_k(x_i) - y_{i,k}$$

where $y_{i,k} = 1$ if and only if x_i belongs to class k. In both cases, we ended up with the learners being fit to the ensemble model's residuals. Intuitively this means the weak learners try to correct the ensemble model's mistakes instead of predicting the true values.

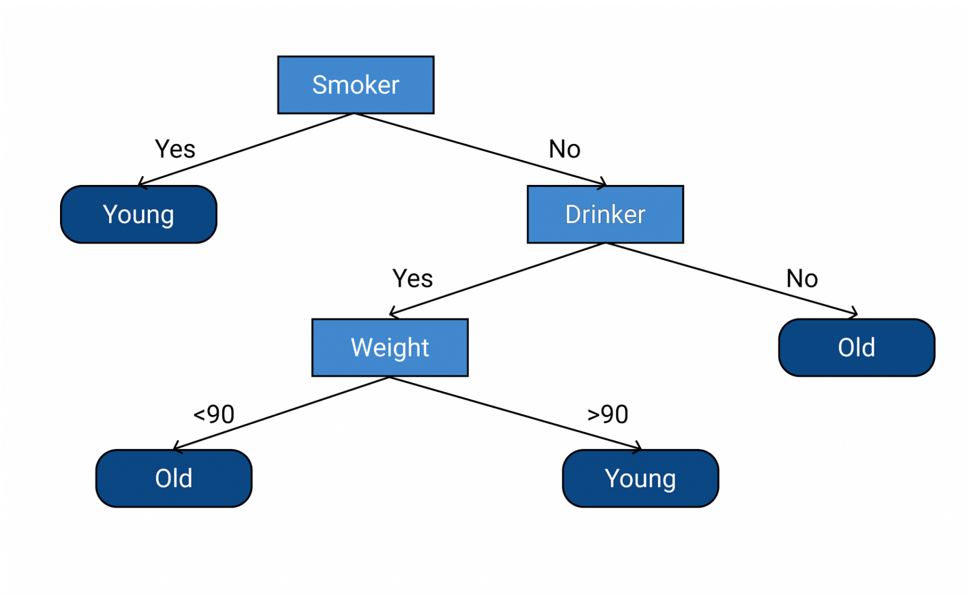
Even though theoretically, we could use a wide variety of machine learning models as the weak learners, in practice, the decision trees are used almost exclusively. The algorithm is then often referred to as gradient boosted trees. In general, a decision tree consists of a root node, a number of interior nodes, and a number of terminal nodes. The root node and interior nodes, referred to collectively as nonterminal nodes, are linked into decision stages; the terminal nodes represent final classifications. [46] The decision trees are grown by recursively selecting a feature and a split point and splitting the current node into two daughter nodes. The procedure stops after a predefined stopping condition is triggered; for example, the tree reaches the maximum depth. Popular methods measuring the quality of a split are based on information gain [44] for classification and squared error for regression.

Algorithm 1 Gradient boosting machine

```
Input:
Training set $\mathcal{T} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\} \subseteq (\mathcal{X} \times \mathcal{Y})^n$
Differentiable loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$
Number of iterations $K \in \mathbb{N}$
Output:
Trained model $h : \mathcal{X} \to \mathcal{Y}$
1: $f_0(x) = \arg\min_\gamma \sum_{i=1}^{n} \ell(y_i, \gamma)$
2: for $k = 1$ to $K$ do
3:     $g_k = \left[ \frac{\partial \ell(y_i, f_k(x_i))}{\partial f_k(x_i)} \right]_{i=1}^{n}$
4:     $\theta_k = \arg\min_\theta = \sum_{i=1}^{m} [(-g_k)_i - b(x_i; \theta)]^2$
5:     $\beta_k = \arg\min_\beta \sum_{i=1}^{m} \ell\left(y_i, f_k(x_i) + \beta b(x_i; \theta_k)\right)$
6:     $f_k(x) = f_{k-1}(x) + \beta_k b(x; \theta_k)$
7: return $h = f_K$
```

4.2.2 Artificial neural networks

Artificial neural network (or simply neural network) models are models inspired by the functioning of the human brain. Recently neural networks



.

Figure 4.2: An example of a decision tree which decides whether the given person will die young or old. Image taken from [3]

have gained huge popularity for achieving great results in almost any field of artificial intelligence. However, the first mathematical models of a neuron were devised as early as 1943 by McCulloch and Pitts [37]. An illustration of the model can be seen in Figure 4.3. The model consists of weights, bias, and the activation function. An example of an activation function is sigmoid: $\varphi(z) = \frac{1}{1+e^{-z}}$. The output is computed by computing a linear combination of the inputs with the weights, adding the bias, and applying the activation function.

A neural network is just a collection of units connected together; the properties of the network are determined by its topology and the properties of the neurons. [42] The neural networks are built by connecting multiple layers together. In Figure 4.4, we can see a neural network with five inputs and five outputs. In between we can see two hidden layers, each with twelve neurons. Notice how every neuron in one layer is connected to every neuron in the next layer. These layers are called fully connected or dense. Each layer performs an affine transformation on its input:

$$y = Ax + b,$$

Where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m\times n}$ and $y, b \in \mathbb{R}^m$. After that, the activation function φ is applied on y. The purpose of the activation function is to introduce non-linearity. Since a composition of affine transformations is also an affine transformation, there would be no advantage in using more than one layer. Moreover, the network would be able to model only linear relationships. There are many different types of layers, such as convolutional or pooling layers.

When computing the output of the network, we can think of the information traveling from the input layer to the output layer through the connection only in one direction (in Figure 4.4 it would be "left to right"). This kind of architecture is called *feedforward* neural network. Neural network architectures that allow the connections between the nodes to form a cycle are called *recurrent* neural networks. In this thesis, we will only talk about the feedforward neural networks.

For the neural network to solve a task, many parameters, for example, the weights in the dense layers, need to be set. To do that, we need a training set and a loss function ℓ. We want the parameters to be set so that the empirical loss on the training dataset is minimized:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta) = \arg\min_{\theta} \frac{1}{M} \sum_{i=1}^{M} \ell(y_i, f_\theta(x_i)),$$

where $f_\theta(x_i)$ denotes the network's output given x_i as the input with θ as the parameters. To do that, we can use the gradient descent algorithm. Thanks to the properties of the derivative, the gradient of $\mathcal{L}$ is the gradient of ℓ averaged over all training samples. In each iteration, the gradient of the loss function with respect to the network's parameters is computed. Then the parameters are moved in the opposite direction. It is necessary that every layer in the network represents a differentiable function. Because the output of the network can be thought of as a composition of multiple functions (one for each layer), the gradient can be computed by repeatedly applying the chain rule:

$$\frac{\partial \ell(f_\theta(x_i))}{\partial \theta} = \frac{\partial \ell(f_\theta(x_i))}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i))}{\partial \theta}.$$

Because $f_\theta(x_i)$ is a composition of multiple functions, the chain rule can be applied again on the term $\frac{\partial f_\theta(x_i))}{\partial \theta}$. This process of computing the gradient is called *backpropagation*. Since most of the training computation time is spent computing the gradient, it is important that the implementation is well optimized. In practice computing, the network's output and the gradient can be realized by performing matrix multiplications. Since this operation can be easily parallelized, being able to run the computation on special hardware such as graphics processing units (GPUs) provides a significant speedup.

Algorithm 2 Gradient descent

```
Input:
Learning rate $\alpha$
Loss function $\mathcal{L}$
Output:
Local optimum $\theta^*$
1: initialize $\theta$
2: while not converged do
3:     $\nabla\mathcal{L} = \frac{\partial \mathcal{L}(\theta)}{\partial \theta}$
4:     $\theta = \theta - \alpha\nabla\mathcal{L}$
5: return $\theta^* = \theta$
```

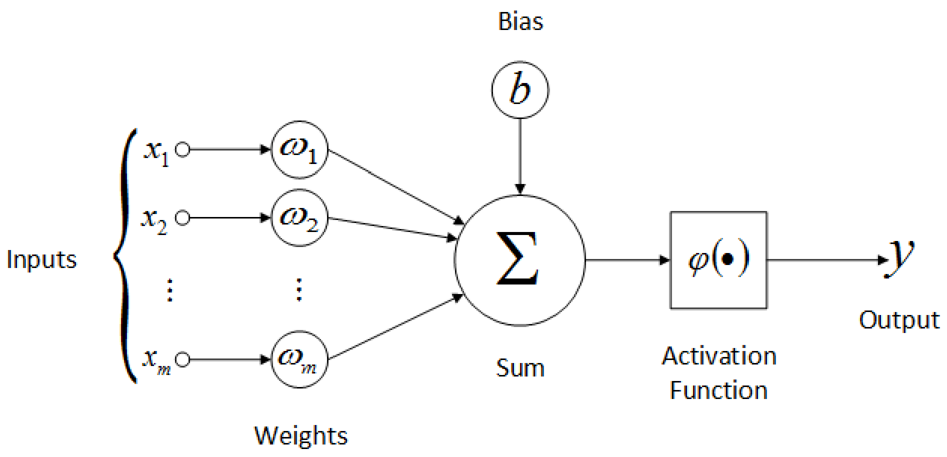


Figure 4.3: Mathematical model of a neuron. It computes $y = \varphi(\sum_{i=1}^{m} x_i\omega_i + b)$. Image taken from [18].

The outline of the algorithm can be seen in the algorithm 2. Notice that in each iteration, the gradient is computed using all samples in the training dataset. This is called the *full-batch* approach. The opposite approach would be to use a subset of the training dataset – the *mini-batch* approach. The modification to the gradient descent using this approach is called stochastic gradient descent (SGD). There are many other different algorithms that aim to optimize the empirical loss, such as Adam [33] or Adadelta [58], but they are all based on optimizing the loss function using the computed gradient.

4.2.3 Graph neural networks

Graph neural network (GNN) is a type of neural network architecture whose input is in the form of a graph. Depending on the architecture, GNNs can work with node features, edge features and weights, and different edge types. Some architectures can even work with hypergraphs – a generalization of graphs where the edges can join an arbitrary number of vertices. [7] Examples of real-life data which can be an input to the graph neural network include models of molecules, social networks, financial transactions, or navigation graphs. The applications include node classification, node representation learning, graph classification, graph generation, and link prediction. [54]

In [54] the graph neural network are divided into five categories:

- Graph Convolution Networks (GCNs) – Generalize the operation of convolution from traditional data (images or grids) to graph data. For a given node, the output of the graph convolution layer depends on the outputs of the previous layer for its neighbors.

- Graph Attention Networks - Similar to GCNs, the difference is they assign larger weights to more important nodes. The weights are learned together with the network's parameters.

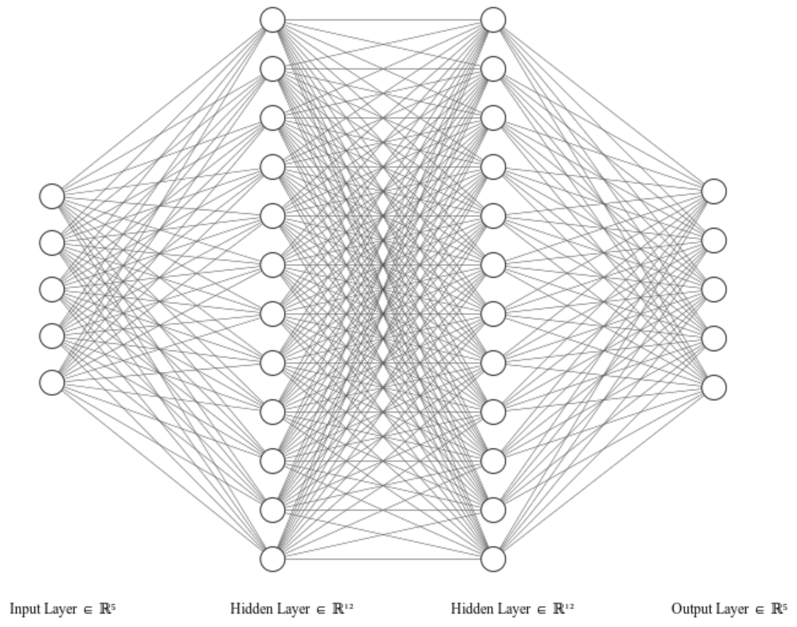


Figure 4.4: A neural network fith five inputs, five outputs and two hidden layers.

- Graph Auto-encoders – Aim to find a low-dimensional representation of the graphs using the auto-encoder architecture.
- Graph Generative Networks – Able to generate plausible graph structures from the given data.
- Graph Spatial-temporal Networks – Aim to learn unseen patterns from spatial-temporal graphs. spatial-temporal graphs are graphs whose features can change over time.

For the rest of this thesis, we will only talk about the graph convolution networks and graph attention networks because we think they are the most relevant for our task. The graph convolution networks can be further divided into two categories:

- Spectral-based – these methods are based on performing the eigen decomposition of the Laplacian matrix[4] of the graph. The convolution is performed as a matrix multiplication in the Fourier domain.
- Spatial-based – these methods work by using the local neighborhoods of the vertices to update the representation vector of each vertex.

The graph convolution networks work by generalizing the convolution operator to work with arbitrary graph structures. The essential idea is to iteratively update the node representations by combining the representations

[4]Laplacian matrix L is equal to $D - A$, where D is the degree matrix and A is the adjacency matrix.

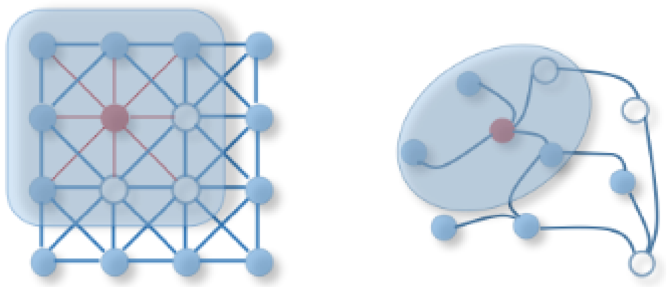


Figure 4.5: Difference between convolution on a grid (left) and a graph (rights). The convolution is computed for the red node. Image taken from [53]

of their neighbors and their own representations. [53] To find a representation of the nodes, multiple graph convolutional layers can be stacked together. There are many proposed graph convolution operators, but most of them can be characterized by the following schema:

$$h_u^{(k+1)} = \text{UPDATE}^{(k)}\left(h_u^{(k)}, \text{AGGREGATE}^{(k)}(\{h_v^{(k)} \mid v \in \mathcal{N}(u)\}\right).$$

Let us look at the equation and explain all the used variables and symbols.

- $h_u^{(k+1)}$ is a representation vector of node u in the (k+1)th layer.
- UPDATE is a function whose input is the representation of the node in the previous layer and aggregated representation of the node neighbors. The output is a new representation of the node.
- AGGREGATE is a function whose input is the representations of a node's neighbors, and the output is their aggregated representation. It should be *permutation invariant* which means that its value does not depend on the order in which the neighbors are processed.[5]
- $\mathcal{N}(u)$ is the set of the neighbors of u.

To summarize, the general operator works by updating the node's representation by using its previous representation and aggregated representation of its neighbors. We can see that if we use only one layer, each node's representation will be influenced by its own and its neighbors' representation. If we add another layer, each node's representation will also be influenced by its neighbors' neighbor's information. Figure 4.6 shows the computation with two layers. If we think that there are long-distance dependencies in our graph, it may be wise to stack many layers on top of each other. On the other hand, if we think that each node is only influenced by its close neighbors, it can be sufficient to only use a few layers. When too many layers are used, each node influences each node, so all the representations may be very similar. This problem is called *over-smoothing* [13].

Now that we have explained the general design of a graph convolution operator, we will describe several existing designs.

[5]This holds implicitly because we defined the domain of the function to be the power set of the graph nodes.

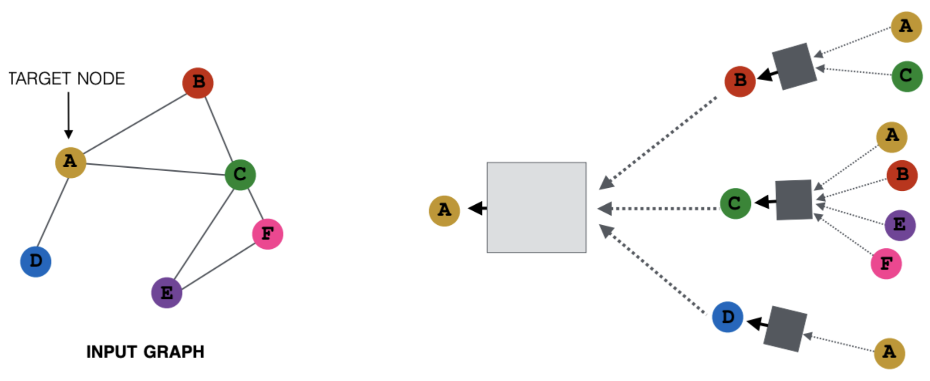


Figure 4.6: Graph convolution example with two convolution layers. A representation of the node A in the input graph (left) is computed by aggregating the representations of its neighbours. Image taken from [2]

GraphConv

Operator introduced in [39], the updated representation is computed as

$$h_u^{(k+1)} = \Theta_1 h_u^{(k)} + \Theta_2 \sum_{v \in \mathcal{N}(u)} e_{v,u} \cdot h_v^{(k)},$$

where Θ_1 and Θ_2 are matrices of learnable parameters and $e_{v,u}$ is the weight of the connection from node v to node u - 1 for unweighted graph. We can clearly see how the computation follows the general schema. The UPDATE function multiplies the current representation by Θ_1 and adds it to the aggregated representation of the node's neighbors. The function AGGREGATE performs the weighted sum of the neighbors' representation and multiplies it by Θ_2. Optionally, the AGGREGATE function can be set to return the weighted mean or maximum.

GATConv

Operator introduced in [49], the updated representation is computed as

$$h_u^{(k+1)} = \alpha_{u,u} \Theta h_u^{(k)} + \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \Theta h_v^{(k)},$$

where the attention coefficients $\alpha_{u,v}$ are computed as

$$\alpha_{u,v} = \frac{\exp\left(\mathrm{LeakyReLU}(a^\top [\Theta u \| \Theta v])\right)}{\sum_{w \in \mathcal{N}(u) \cup \{u\}} \exp\left(\mathrm{LeakyReLU}(a^\top [\Theta u \| \Theta w])\right)}.$$

a is a vector of learnable parameters, Θ is a matrix of learnable parameters, and $\|$ represents the concatenation of two vectors. The function LeakyReLU is defined as

$$\text{LeakyReLU}(x) = \begin{cases} x & x > 0 \\ 0.01x & \text{otherwise.} \end{cases}$$

GINConv

Operator introduced in [55], the updated representation is computed as

$$h_u^{(k+1)} = h_\Theta \left((1+\epsilon) \cdot h_u^{(k)} + \sum_{v \in \mathcal{N}(u)} h_u^{(k)} \right),$$

where h_Θ denotes a neural network, for example, a multi-layer perceptron, and ϵ is a given constant or learnable parameter.

GCNConv

Operator introduced in [34], the updated representation is computed as

$$h_u^{(k+1)} = \Theta^\top \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{e_{v,u}}{\sqrt{\hat{d}_v \hat{d}_u}} h_v^{(k)},$$

where $\hat{d}_u = 1 + \sum_{v \in \mathcal{N}(u)} e_{v,u}$, where $e_{v,u}$ denotes the edge weight from node v to node u - 1 for unweighted graph. Θ is a matrix of learnable parameters.

4.3 Metrics

To compare and report the performance of the models, we need suitable metrics for our task. In this section, we will define the metrics that will be used in the later chapters of the thesis. The vectors y and $\hat{y}$ will represent the true and predicted labels respectively.

4.3.1 Regression metrics

Regression metrics measure the difference between two vectors of values. They penalize the model based on the distance between the true and predicted values. They can be used in ordinal regression to measure how close are the predicted labels to the true ones.

The first metric we will use is called root-mean-square error (RMSE). The metric is sensitive to outliers because it is based on computing the squared differences between the values. It is computed as:

$$\text{RMSE}(y, \hat{y}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2}.$$

4.3.2 Classification metrics

Classification metrics measure the model's ability to predict a correct label from the finite set of labels. They treat the set of labels as a nominal quantity – they do not take into account a possible ordering of the labels.

The most easily interpretable metric is accuracy. It is computed as the proportion of correctly classified samples:

$$\text{Accuracy}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} [\![y_i = \hat{y}_i]\!]^6$$

The accuracy can be misleading when the labels in the dataset are not represented equally. There are datasets where this is the case, for example, in fraud detection. A model predicting all transactions to be non-fraudulent will achieve a very good accuracy score while being useless.

A more suitable metric is the F_1 score. It is defined for binary classification but can be extended for multi-class classification as well. We will define it in terms of precision and recall. In the binary classification setting, the samples are often labeled as positive and negative, which means that $\mathcal{Y} = \{+1, -1\}$. When a model predicts a sample's label, there are four different outcomes, two in which the model is right and two in which the model is not. When we count the number of the outcomes, we end up with the *confusion matrix*. This matrix can be seen in the following table. We will define the metrics in terms of the values in the confusion matrix instead of as functions of y and $\hat{y}$, which is standard in literature and makes the definitions more readable.

$y/\hat{y}$	+1	-1
+1	TP	FN
-1	FP	TN

Table 4.1: Different outcomes in binary classification.

Precision measures what portion of samples the model predicted to be positive is actually positive:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

Recall measures what portion of actual positive samples the model predicted to be positive:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

The F_1 score is computed as

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

[6] $[\![\cdot]\!]$ is called *Iverson bracket* and it evaluates to 1 if the statement inside the brackets is true and 0 otherwise.

Its value is in the range $[0, 1]$, where the model assigning all labels correctly will have a score of 1.

When we are dealing with multi-class classification, we can approach it as k binary classifications - for each label, we divide the samples into those which are assigned the label and those who are not. This is called the one-vs-all approach. Then we can calculate the F_1 score for each label. There are two ways to combine the scores into one score:

- Weighted – the final score is the weighted mean of the individual scores, where the weights correspond to the number of samples with the corresponding label.
- Macro – the final score is the arithmetic mean of the individual scores.

The first approach favors the classes which occur more frequently in the data. The second approach gives the same importance to all classes. It is, therefore, more suitable for imbalanced datasets. Another approach is to compute the sum of the confusion matrices and compute the F_1 score as we would for binary classification. This approach is called micro-averaging, and similarly to the weighted score, it also favors the more frequent classes.

Chapter 5

Data and preprocessing

This chapter will focus on the data which we worked with in this thesis. The data consists of a navigation graph G and a set of user-generated trips T. We created a data processing pipeline, which used this input data to create a navigation graph enriched with features extracted from the trips. This enriched navigation graph will be used as an input of the machine learning methods described in Chapter 4.

We will begin by briefly describing the steps of the data processing pipeline. Then we will describe the format and contents of the used data (5.2). After that, we will describe the individual steps of the data-processing pipeline (5.3). Finally, we will provide the analysis of the final dataset (5.4).

5.1 Overview of the data processing pipeline

The diagram of the procedure can be seen in Figure 5.1. The procedure begins by combining the GPS measurements with their matching and computing the features of each segment. By a segment, we mean one particular traversal on a particular edge. Because there can be some nonsensical values in the measurements, some segments are filtered out. After that, several features are extracted both from the navigation graph and the trip data and added to the existing edge features of the navigation graph. Finally, some edges are removed from the navigation graph based on the extracted feature in the previous step. The output of the pipeline is the subgraph of the original navigation graph, whose edges were enriched by several extracted features. The edges, which do not appear in any segment, i.e., they were never visited, are removed because no features can be extracted for them. This should not be a problem because the trips cover the majority of the original navigation graph. Finally, the graph is transformed into the so-called edge-based representation. This representation is based on transforming the given graph so that the edges in the original graph are represented as the nodes in the new graph, and all connections in the original graph are preserved. This way, we can perform edge classification on the original graph using models designed for node classification.

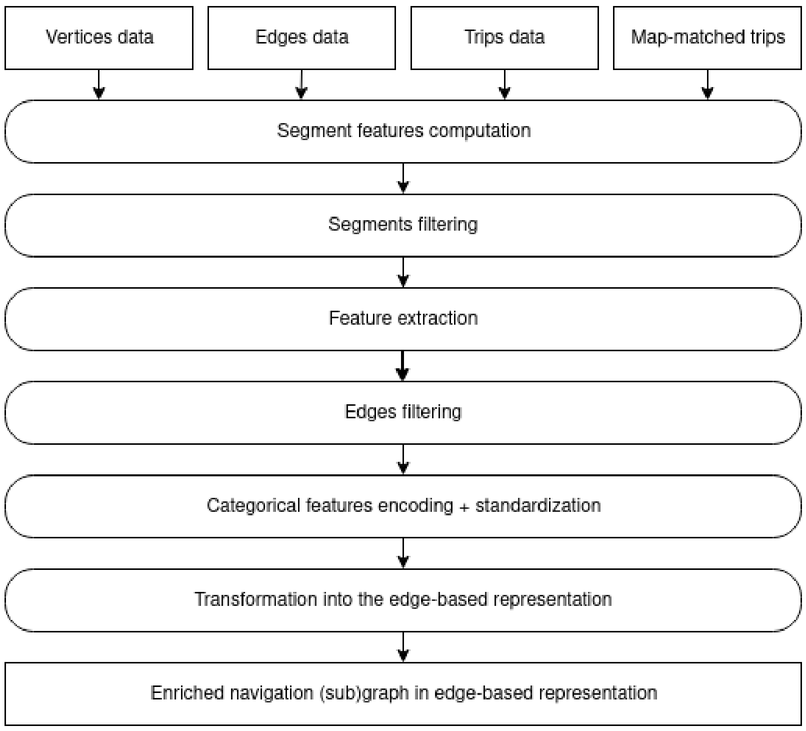

Vertices data
Edges data
Trips data
Map-matched trips
Segment features computation
Segments filtering
Feature extraction
Edges filtering
Categorical features encoding + standardization
Transformation into the edge-based representation
Enriched navigation (sub)graph in edge-based representation

Figure 5.1: The data-processing pipeline

5.2 Data sources description

The data was compiled using four data sources:

1. Vertices data – information about the vertices in the navigation graph
2. Edges data – information about the edges in the navigation graph
3. Trips data – raw GPS measurements from the users
4. Map-matched trips – the sequences of the nodes representing each trip as a path in the navigation graph

Each of the data sources will be described in this section.

5.2.1 Vertices data

The vertices data are stored as a csv[1] file. Each line of the file represents one vertex of the navigation graph and it contains the vertex features separated by a comma. The vertices have the following features:

- Node id – a unique identifier of the nodes
- Latitude – latitude of the node's location

[1]comma-separated values

- Longitude – longitude of the node's location
- Elevation in meters – elevation of the vertex above the sea level

5.2.2 Edges data

The edges data are stored in the same format as the vertices data. Each line represents one edge of the navigation graph. The features of the edges are:

- From id – id of edge's starting node
- To id – id of the edge's ending node
- Number of lanes – the number of car lanes on the road in the edge's direction.
- Maximum allowed speed in km/h
- Cycle infrastructure – a categorical value representing the type of cycling infrastructure on the road. Possible values are:
  - TRACK – dedicated track for cyclists separated from car traffic.
  - ZONE – pedestrian zone with allowed bike access.
  - LANE – lane dedicated for cyclists.
  - SHARROW – advisory cycle lane.
  - NONE – no cycling infrastructure present.
- Road type – a categorical value representing the type of the road. Possible values are:
  - PRIMARY – I. class roads.
  - SECONDARY – II. class roads.
  - TERTIARY – III. class roads.
  - SERVICE – generally for access to a building, service station, industrial estate, etc., or unclassified roads connecting houses and buildings.
  - RESIDENTIAL – living streets (streets where the maximum allowed speed is lowered due to contact with pedestrians).
  - OFFROAD – roads in the countryside, mostly unpaved.
  - FOOTWAY – a walk for pedestrians.
  - CROSSING – a pedestrian crossing - a place designated for pedestrians to cross the road.
  - STEPS – stairs.
  - CYCLEWAY – dedicated road/street for cyclists.
  - UNKNOWN
- Paved – a boolean value signaling that the road has a paved surface.

- No entry – a boolean value signaling that it is prohibited to travel on this edge, typically the opposite direction to a one-way street.

- Surface – a categorical value representing information about the quality of the road's surface. This is the attribute that we want to correct. Possible values are:

  - UNKNOWN,
  - EXCELLENT – e.g. brand new asphalt.
  - GOOD – e.g. old asphalt roads or concrete roads, or paving stones with very narrow gaps.
  - INTERMEDIATE – a bit more rugged surface, compacted unpaved roads.
  - BAD – e.g. cobblestones with bigger gaps, not compacted unpaved roads.
  - HORRIBLE – roads hardly used for a bicycle.
  - IMPASSABLE – roads that should be used only as a last resort.

  UNKNOWN and IMPASSABLE edges are very rare in the data, so we decided not to include the edges with these labels.

- OSM tags – the attribute Surface is inferred from the following three categorical features from Open Street Maps. For this reason, these features cannot be in the input of the models. Moreover, these features are often missing from the data. When none of the tags are present, the Surface attribute is derived using simple rules such as "(Road type = FOOTWAY) $\implies$ (Surface = INTERMEDIATE)".

  - OSM smoothness[2] – classifies physical usability of the edge for wheeled vehicles, particularly regarding surface regularity/flatness, it corresponds almost exactly to the feature Surface, an example of possible values are excellent, intermediate, horrible
  - OSM tracktype[3] – measures how well maintained the edge is, possible values are grade1, grade2, ..., grade5
  - OSM surface[4] – the type of the surface, example of possible values are: concrete, cobblestone or grass

The vertices data and edges data together form a navigation graph, which follows the definition from 3.1.1.

[2]https://wiki.openstreetmap.org/wiki/Key:smoothness
[3]https://wiki.openstreetmap.org/wiki/Key:tracktype
[4]https://wiki.openstreetmap.org/wiki/Key:surface

5.2.3 Trip data

The set of trips T was gathered by around four thousand volunteers, who rode their bikes with the provided smartphone app turned on. The app periodically measured their position as well as velocity using GPS. The trip data are stored as a collection of files in the GPX format, where each file represents one trip. An example of a GPX file can be seen in Figure 5.2. We will now describe the contents of the GPX files, which we used to create our dataset.

User and trip id

This information was not in the GPX file itself but was contained in the name of the files. The name of each file consisted of a unique id of the user that made the trip and a unique id of the trip. This means that even though it is not possible to identify the user in real life, it is possible to figure out which trips were made by the same user. The trip id was used to find the file containing the corresponding map-matching.

Bike type

The type of bike that was used to make the trip was in the `<type>` tag. The possible values for the bicycle type are:

- `city_bike` – a bicycle meant to be used to travel short distances in the city, often with limited suspension and transmission
- `road_bike` – a bicycle with thin tires used to travel exclusively on the road
- `mountain_bike` – a bicycle with good suspension and wide tires, can be ridden off-road without big discomfort
- `hybrid_bike` – something between a road bike and mountain bike, can be used to travel both on and off the road
- `folding_bike` – a bicycle with the ability to be folded and carried around, mainly used to combine with the public transport
- `electric_bike` – any type of bicycle with a battery and electric motor, which helps the user to ride with less effort
- `cargo_bike` – a bicycle used to transport cargo, for example, small packages or meals to be delivered
- `fixie_bike` – a bicycle without the freewheel mechanism, often with one gear ratio
- `other` – any other type of bicycle which does not fall into any of the previous categories

```
<?xml version="1.0" encoding="UTF-8" ?>
<gpx xmlns="http://www.topografix.com/GPX/1/1" version="1.1" creator="Cyclers"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/1/1/gpx.xsd ">
<trk>
<type>mountain_bike</type><trkseg>
<trkpt lat="50.764585" lon="15.048591">
        <ele>435</ele>
        <time>1970-01-01T13:00:00Z</time>
        <speed>2.694141</speed>
        <accuracy>16.885000</accuracy>
        <bearing>240.808090</bearing>
</trkpt>
...
<trkpt lat="50.762488" lon="15.041842">
        <ele>410</ele>
        <time>1970-01-01T13:03:03Z</time>
        <speed>3.664586</speed>
        <accuracy>17.171000</accuracy>
        <bearing>168.328430</bearing>
</trkpt>
</trkseg>
</trk>
</gpx>
```

Figure 5.2: GPX format

- `unknown` – the user has not filled in the information about the bicycle type

GPS measurements

The most important part of the GPX file is the list of the GPS measurements. The GPS measurements following the definition from 3.1.1 can be found in the `<trkpt>` tags. Each measurement contains:

1. Latitude, longitude and elevation – the geographical measurements
2. Time – the timestamp of the measurement. To anonymize the users, the provided timestamps do not contain the exact time the user started the trip. Instead, the first timestamp is rounded down to the whole hours, and the following timestamps represent the relative time change. For example, instead of having timestamps (12:38, 12:41, 12:43), we have (12:00, 12:03, 12:05). This way, it would be harder to identify a specific user based on their habits. Even though this results in a slight information loss, we do not suppose the exact starting time to the minutes significantly affects the ride.
3. Accuracy – the current accuracy of the geographical measurements in meters, based on the GPS signal strength
4. Bearing – the direction of the front wheel in degrees

5.2.4 Map-matched trips

The GPS measurements alone do not represent the movement of the user in the navigation graph. To do that we need a function f_G, which for each trip t assigns a sequence of vertices $(v_1, v_2, \ldots, v_{n_t})$, such that

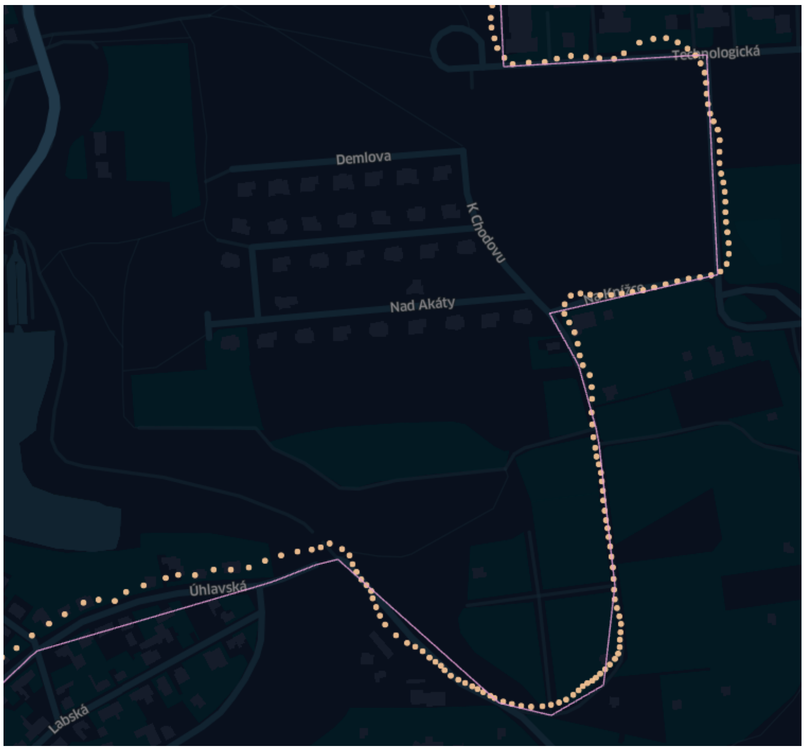


Figure 5.3: The GPS measurements (yellow dots) and the corresponding path (pink lines)

$$\forall i \in \{1, 2, \dots, n_{n_t} - 1\} : (v_i, v_{i+1}) \in E(G).$$

This function performs the so-called *map matching*. The desired property of the function f_G is to map the trip to the sequence so that the positions of the GPS measurements are close to the path defined by the sequence. Designing nor implementing a map matching algorithm is not the topic of this thesis, and our data had already been map-matched. An example of a scalable map-matching algorithm was introduced in [21].

Each map-matched trip was stored as one file containing the list of node ids – the output of the map matching function. Just like the files containing the GPS measurements, the name of each file contains the id of the trip that it represents and the id of the user. An example of a map-matched trip together with the corresponding measurements can be seen in Figure 5.3.

5.3 Data processing

This section will describe the steps which the data processing pipeline performed in order to create the final dataset. The steps could be seen in Figure 6.1 as the round boxes.

5.3.1 Segment features computation

A segment is one particular traversal of an edge. Using the GPS measurements, we computed two features of each segment.

Average velocity

For each trip, we computed the average travel velocity on the edges, which the user traveled on. Given a trip $t = (m_1, m_2, \ldots, m_n)$ and its matching $v = (v_1, v_2, \ldots, v_m)$, the average velocity of a segment (v_i, v_{i+1}) was computed as the average of the velocities of the measurements taken in between entering and leaving the edge (v_i, v_{i+1}). The first respectively the last measurement in the segment is the one closest to the vertex v_i respectively v_{i+1}.

$$j = \min_{j'} d(v_i, m_{j'}),$$

$$k = \min_{j'} d(v_{i+1}, m_{j'}),$$

$$\text{velocity}_t((v_i, v_{i+1})) = \frac{1}{k - j + 1} \sum_{l=j}^{k} v(m_l),$$

where $d(v, m)$ denotes the distance between a vertex v and a measurement m, and $v(m)$ denotes the measured velocity in of the measurement m.

Bearing swing

For each trip, we computed a metric we call *bearing swing*. The metric is computed similarly to the average velocity. We begin by computing the indices of the measurements closest to the starting and ending points of the edge – j and k. Then we compute the bearing swing as the standard deviation of the differences in the bearing of consecutive measurements:

$$B = \{b(m_{l+1}) - b(m_l) \mid l = j, j + 1, \ldots, k - 1\},$$

$$\text{swing}_t((v_i, v_{i+1})) = \sqrt{\frac{1}{|B|} \sum_{b \in B} (b - \bar{b})^2},$$

where $b(m)$ denotes bearing of the measurement m and $\bar{b}$ is the average element of the set B. The aim of the metric is to detect when the user makes large steering adjustments. Note that when the user makes smooth maneuvers, for example, in turns, this metric stays low.

5.3.2 Segments filtering

There were two problems with the user-generated data. The first problem was caused by the measuring devices failing to take a measurement for a long period of time, and the second problem was suspicious values in some measurements.

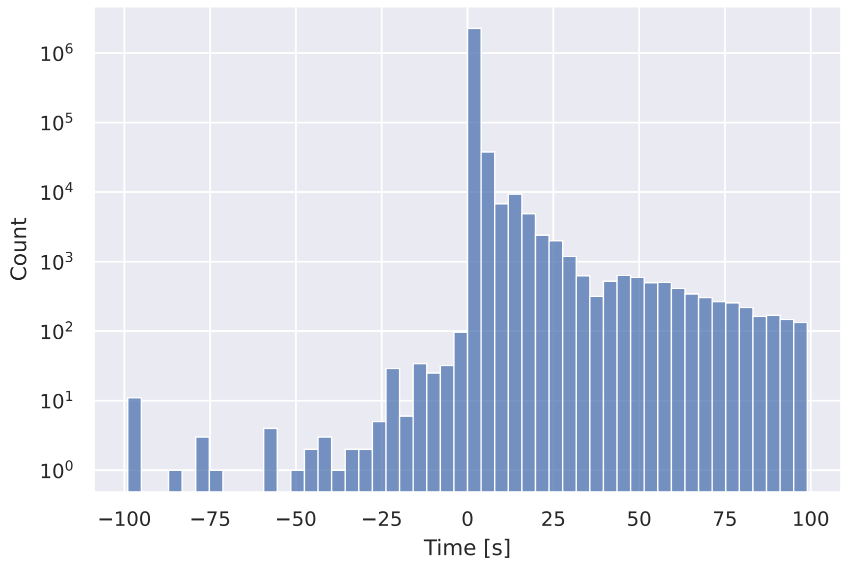


Figure 5.4: The differences between consecutive GPS measurements (cut at 100 for visualisation purposes). The y-axis is in log scale.

GPS malfunctions

During the travel, it can happen that the smartphone fails to make the GPS measurements for some period of time. For our purpose, it is crucial to have accurate measurements. For example, the traveling velocity on a bike can change drastically in tens of seconds. For this reason, we tried to detect the segments of the trip where no measurement was taken for a longer period of time. The measurements after GPS failure often have the velocity set to zero, indicating an error. However, we do not want to filter out all measurements with zero velocity, as it is possible they do not indicate GPS failure, and the user was forced to stop by the traffic.

Figure 5.4 shows the histogram of differences between consecutive measurements. We can see in the vast majority of times (more than 98% to be more precise), the next measurement is taken within the next five seconds. However, there are also some much larger differences and even negative numbers. Both of these phenomena are most probably the result of some error, and we do not want these measurements in our data.

To cope with this problem, we marked the measurements more than five seconds apart as corrupt and deleted the parts of the map-matched sequence to which these corrupt measurements belonged. The result can be seen in Figure 5.6. We can see there are three gaps among the measured points. The segments in the gaps were filtered out. It may seem we lost a lot of data, but these malfunctions are relatively rare, and this way, we extracted at least some data that would otherwise be lost had we discarded the whole trip. Figure 5.5 shows the histogram of velocities before and after doing this

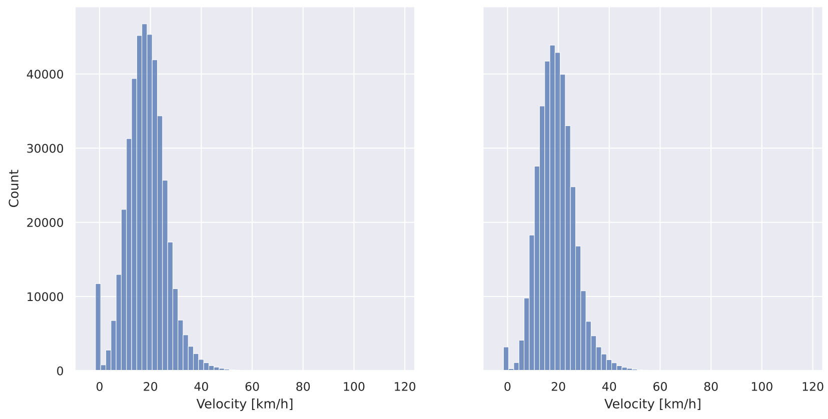


Figure 5.5: Histogram of segment velocities before and after filtering GPS malfunctions

operation. We see that the number of segments with zero velocity decreased to about a fourth, but there is still a noticeable spike around zero. Either we did not manage to detect all GPS malfunctions, or the remaining zero values are caused by a different kind of error.

Velocity

When we look at the histogram of segment velocities in the aforementioned figure 5.5, we see two problems:

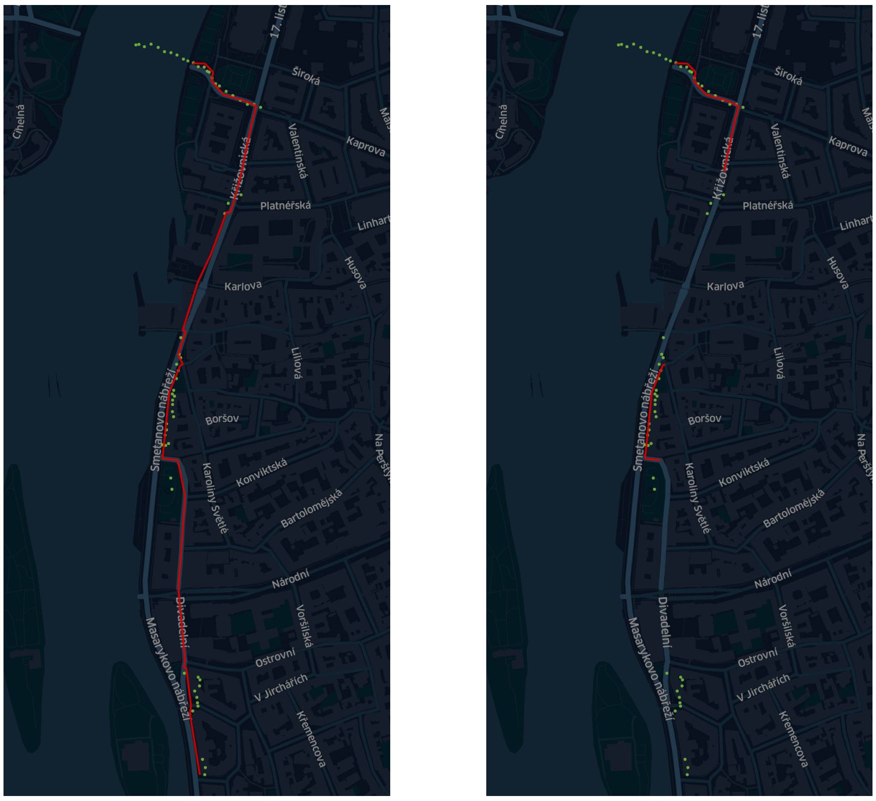
1. No segment should have zero velocity as it is impossible to travel non-zero distance with zero velocity.

2. There is a very low number of very high velocities. Even though professional athletes are able to reach these velocities, it is very probable that these measurements are the result of either an error or a user forgetting to turn the app off and traveling by a vehicle.

We solved both of these problems by filtering out the velocities outside the $(0, 50]$ interval. The resulting histogram can be seen in Figure 5.7. The distribution looks similar to the normal distribution but slightly skewed right.

5.3.3 Feature extraction

We created the dataset by joining the navigation graph and the user-generated trips together. This was done by introducing several new features from the trips and adding them to the features described in this chapter. The added features can be classified into three main categories.



(a) : Before filtering **(b) :** After filtering

Figure 5.6: Detection and filtering of large distance between GPS measurements

Elevation angle

We added one feature that can be computed from the navigation graph alone without the generated trips. The feature is `elevation angle`, and it is an estimate of the angle of the road between two vertices. This feature is very important in the context of bike riding as the relationship between the travel velocity and the elevation angle is very strong. However, there are other factors impacting the velocity, such as the surface quality. By adding this feature to the data, we will make it easier for the models to decide whether the users traveled on the given edge slowly because of a bad surface or because of steep elevation and vice versa. Given two vertices v_1 and v_2, the elevation angle can be computed as:

$$\alpha(v_1, v_2) = \arctan\left(\frac{e(v_2) - e(v_1)}{d(v_1, v_2)}\right),$$

where $e(v)$ denotes the elevation in meters of the vertex v and $d(v_1, v_2)$ denotes the distance between the nodes v_1 and v_2.

Velocity-based features

These features represent how fast the users traveled on the particular edge.

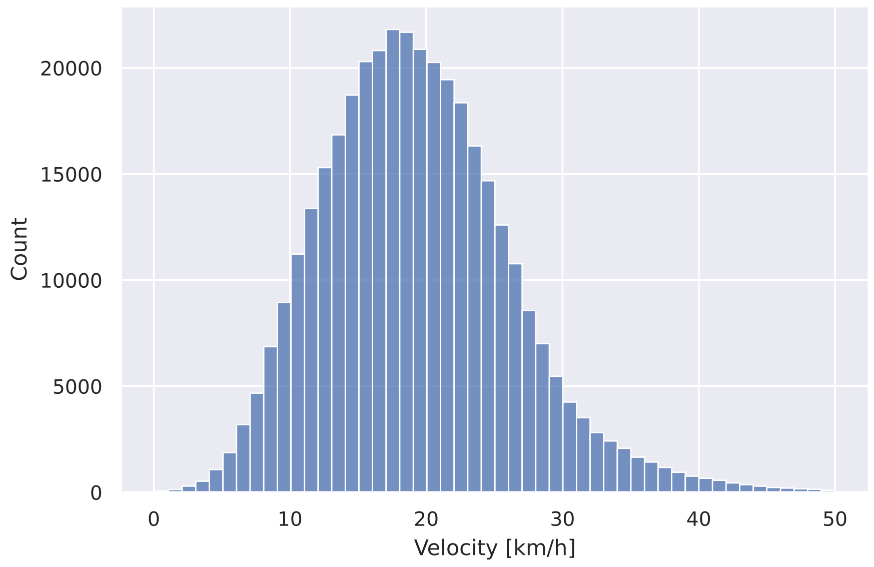


Figure 5.7: Histogram of velocities after filtering

- `Avg_velocity` – the average velocity of the users going through the edge. It is computed as the mean of the segment velocities computed in 5.3.1.
- `Avg_velocity_road_bike` – the average velocity of users using the road bike.
- `Avg_velocity_mountain_bike` – the average velocity of users using the mountain bike.
- `Avg_velocity_other_bike` – the average velocity of users using the rest of the bike types.

The velocities grouped by the bike types were added based on our hypothesis that the velocities of the users of the road bike and mountain bikes are quite different, especially on some surfaces. It is possible that some edges were not traveled on by any user of the given bike type and the average had to be computed from the empty set, resulting in `NaN` results. In that case, we imputed the missing values by the average of the feature across all edges.

User velocity-based features

It is possible that some edges are traveled on by relatively faster riders. To compensate for this phenomenon, we devised features that quantify the answer to the question: "How fast are the users that travel on this edge, relative to the rest of the population?" We start by computing three score metrics for each user:

- `User_Velocity_score_coef` – the user's velocity coefficient, which is computed as their velocity divided by the average velocity on the given edge, averaged over all edges the user traveled on. For example, the value of 1.1 indicates that the user is on average 10% faster than the average user.

- `User_Velocity_score_num_deviations` – the number of standard deviations of the velocities the user's velocity on the given edge is higher (positive number) or lower (negative number), averaged over all edges the user traveled on.

- `User_Velocity_score_percentile` – the user's velocity percentile among the other users' velocities on the given edge, averaged over all edges the user traveled on. Unlike the two previous metrics, it does not work with the statistical properties of the velocities but with the relative ranking of the users.

To compute the features of the given edge, we averaged the previously described score metrics of all users that traveled on the edge.

Average bearing swing

For each edge, we computed the average bearing swing, which we computed by averaging the bearing swings of all segments belonging to the edge, which we computed in 5.3.1. The idea behind this feature is that on the bad surfaces, the users will have to make more sudden movements to select the best trail by, for example, avoiding the potholes.

Most used bike

For each edge, the most used bike on the edge is computed. We think that this information can help the model with the surface quality assessment. If, for example, the most used bike on the edge is a road bike, it may mean that the surface on the edge is reasonably good.

5.3.4 Edges filtering

We spotted some suspicious values in the elevation angles between the nodes. Figure 5.8 shows the histograms of elevation angles before and after filtering. We decided to filter out angles outside the $[-0.15, 0.15]$ interval. Elevation values higher than 12% (roughly 0.12 radians) are considered dangerous, so values such as 1 are absurd. We can see that even after filtering, the borders of the interval contain very few samples. We suppose that the extreme values of the elevation angle are caused by inaccuracies in the measured elevations, as relatively small inaccuracy can impact the elevation angle value drastically, especially on short edges.

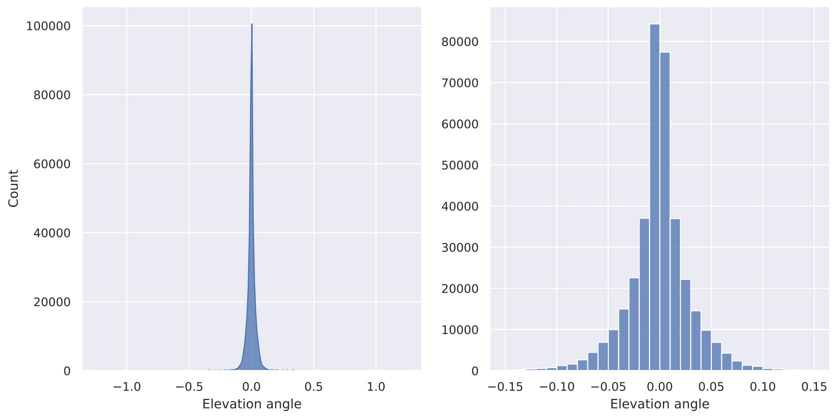


Figure 5.8: Histograms of elevation angles before (left) and after (right) filtering. The values are in radians.

5.3.5 Encoding of categorical features

Many edge features are categorical, but most machine learning models can only work with numerical features. We used the most common encoding of the categorical features called *one hot encoding*. For each categorical feature with k possible values, we introduced k new features. The sample having the ith value of the feature is then encoded by setting the ith new feature to 1 and the rest to 0.

5.3.6 Standardization

Some models, such as SVMs or gradient descent-based models, are sensitive to features having different scales. For this reason, it is beneficial to *standardize* the feature values before applying the algorithms. We standardized the feature values to have zero mean and unit variance by the following transformation:

$$x' = \frac{x - \bar{x}}{s},$$

where $\bar{x}$ is the sample mean, and s is the sample standard deviation of each feature's values.

5.3.7 Transformation into the edge-based graph representation

Because some models are designed to predict the properties of the nodes of the graph rather than the edges, it is beneficial to create the edge-based representation of our graph. Given the graph $G = (V, E)$, its edge-based representation $G' = (V', E')$ is constructed as follows:

$$V' = E$$

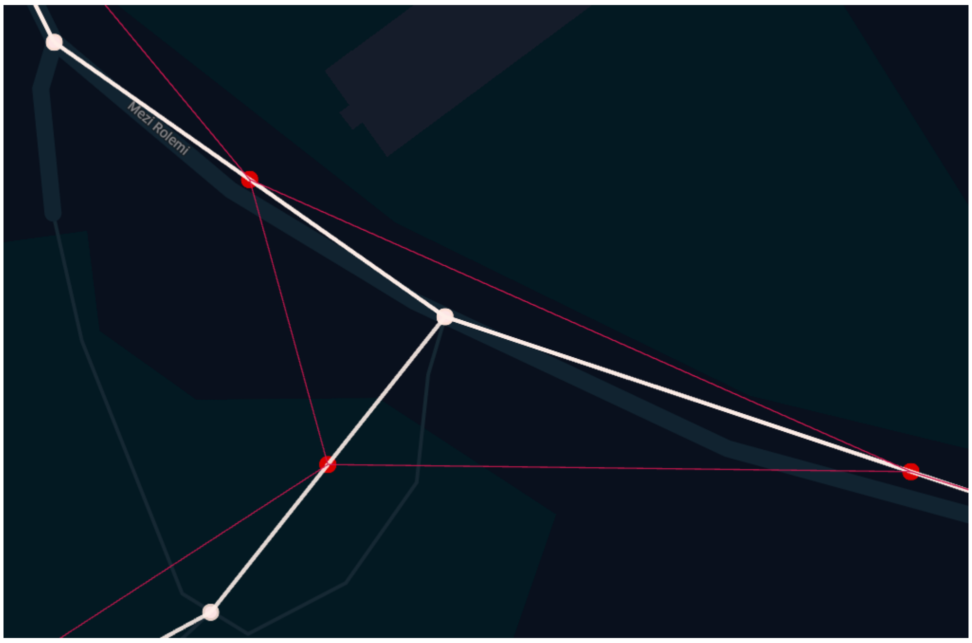


Figure 5.9: The edge based representation (red) of a junction in the original graph (white).

$$E' = \{((v_1, v_2), (v_2, v_3)) \mid (v_1, v_2), (v_2, v_3) \in E\},$$

i.e., the nodes in the new graph are the edges in the original graph, and the edges in the new graph connect the nodes that represent the edges that are connected in the original graph. This can be seen more clearly in Figure 5.9.

In the context of the navigation graph, the features of the vertices in the new graph are the features of the edges in the original graph, there are no edge features in the new graph, and the vertex features from the original graph are lost. In our case, this is not a problem because the vertex features do not contain any useful information.

5.4 Data analysis

The previous section dealt with the process of creating the dataset from the provided data sources. In this section, we will analyze the output of the data processing pipeline. Our dataset can be divided into three parts covering three locations: Prague and surrounding areas, the Jizera Mountains, and the Beskid Mountains. The corresponding subgraphs can be seen in Figures 5.10, 5.11 and 5.12. Table 5.1 shows the size of the final dataset. Even though the data from Prague covers the smallest area of the three, the graph is much denser, and for this reason, it has much more edges. Fortunately, the number of trips made in that area is also much higher, so the features we from 5.3.3 were not computed using a lower number of trips.

In the filtering steps of the pipeline, around 0.8% of the segments and 0.5% of the edges were filtered out. The number of nodes in the edge-based

	Prague	Jizera Mountains	Beskid Mountains
Number of trips	33,096	3,757	5,962
Average trip length	7.6 km	7.5 km	8.1 km
Number of edges	212,653	30,992	43,831

Table 5.1: Statistics of the dataset

representation of the graph was roughly 2.5 times the number of edges in the original graph. In the following subsections, we will analyze the main attributes of the dataset.

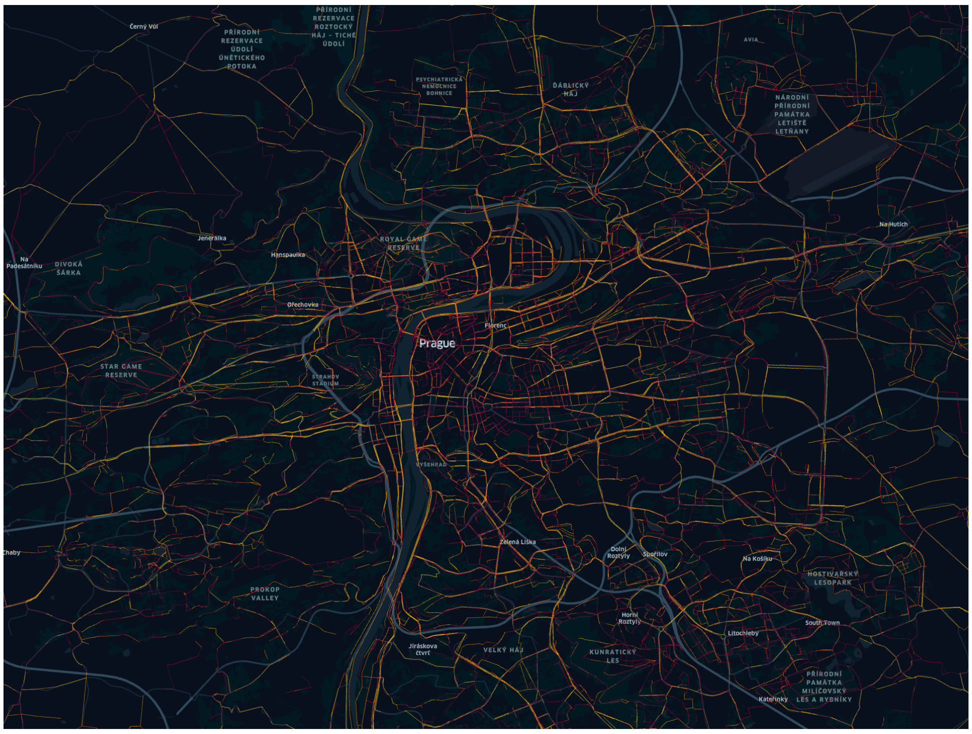


Figure 5.10: Trips from Prague visualised. Brighter color indicates faster average travel velocity.

5.4.1 Edge features

Figure 5.13 shows the histogram of the surface quality labels. We can see that the labels are distributed very unevenly, with the majority of the edges having either EXCELLENT or INTERMEDIATE labels. Figure 5.14 shows the histograms of the main features of the edges. Note that some plots have a log-scaled y-axis so that all values can be seen. We can see that the majority of the edges are paved and have NONE cycle infrastructure. Figure 5.15 shows the Venn diagram of the edges with respect to which OSM tag they have filled in. We can see that more than half of the edges have the surface tag while only about 4.8% have the smoothness tag and about 5.5% have the tracktype tag. Figure 5.16 shows the histogram of the average velocity scores

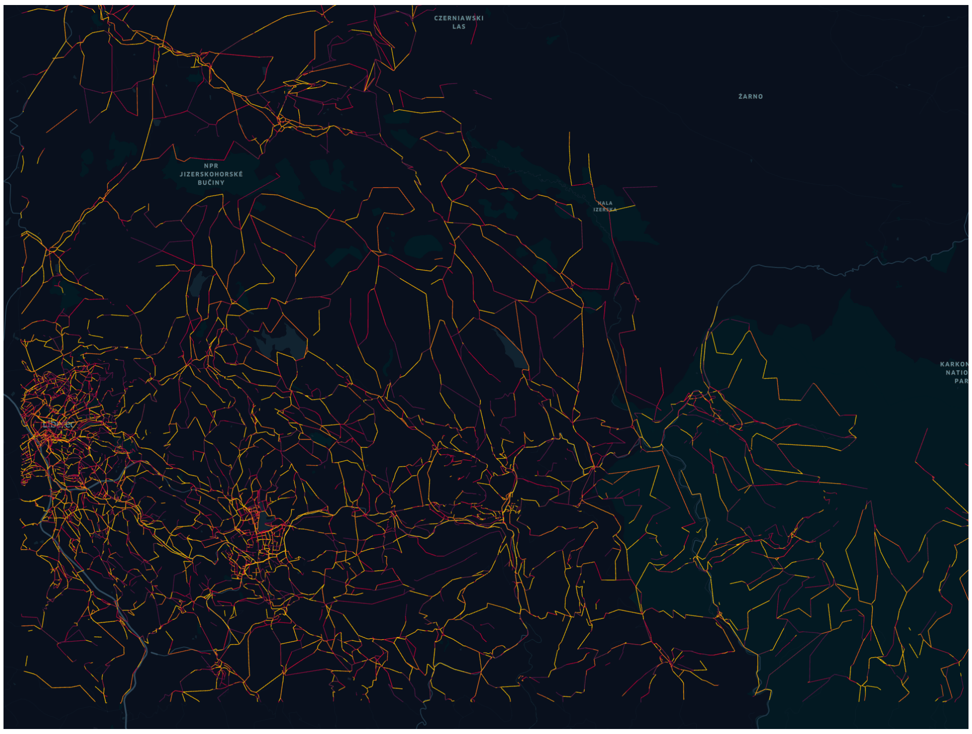


Figure 5.11: Data from the Jizera Mountains

on the edges. We can see that on some edges, there are users traveling 20% faster on average. Figure 5.17 shows the histogram of the number the edges that were visited. We can see that the majority of the edges were visited less than 100 times, but some edges were visited as many as 1400 times.

5.4.2 Segment features

Figure 5.18 shows the histograms of the measured velocities and bearing swings of the segments. We can see, that the average velocity is around 18 km/h and most of the segments have a very low bearing swing.

5.4.3 Relationships between features

Because we will be trying to model the surface quality, it may be useful to analyze its relationship with the other features. Figure 5.20 boxplots of the relationship of the surface quality with the velocity and surface quality. We can see that the worse the surface, the lower the velocity, which is expected. For the bearing swing, the relationship is not this simple, but still, there are differences among the different surfaces. In both cases, there are many outliers, which indicates the influence of other features. Figure 5.22 shows the relationship between the surface quality, the bike type, and the velocity. We can see that the road bikes are the fastest on most surface qualities. We would expect that the mountain bikes would be noticeably faster on low-quality surfaces, but this seems not to be the case. The fixie bike is by far the fastest

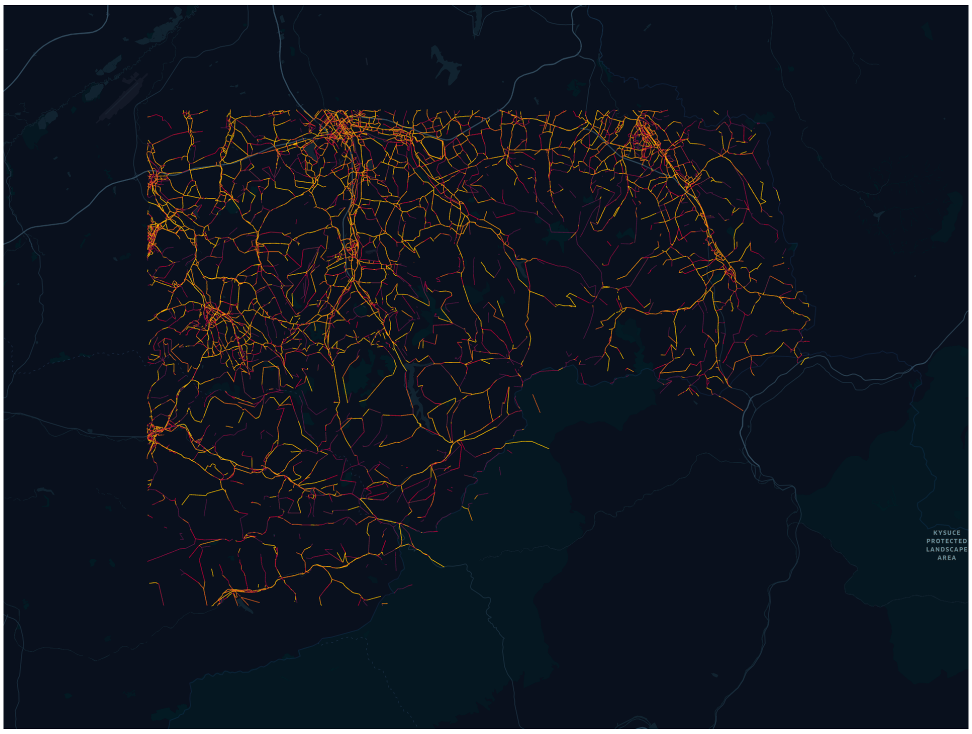


Figure 5.12: Data from the Beskid Mountains

on the good surface, but as we can see in Figure 5.19, this type of bike is very rare in the data, so this result is not significant.

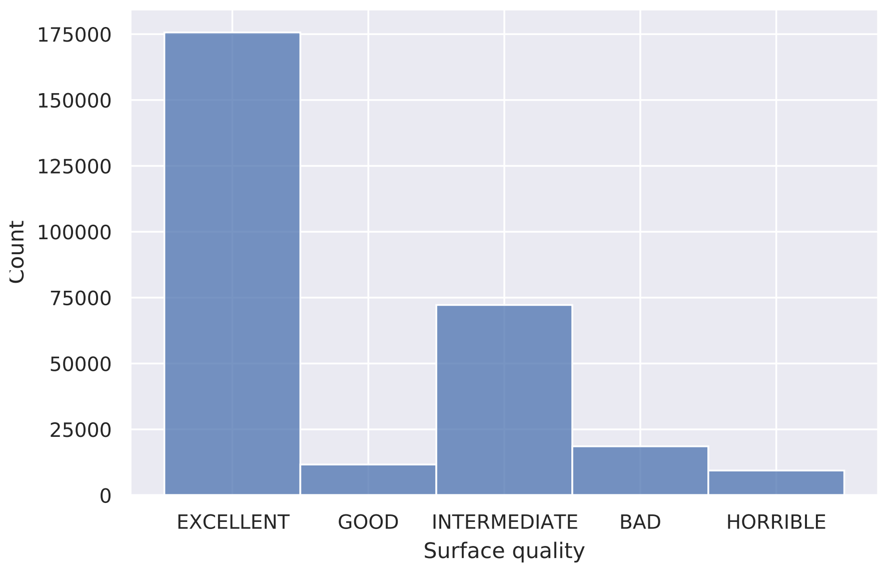


Figure 5.13: Histogram of surface quality types

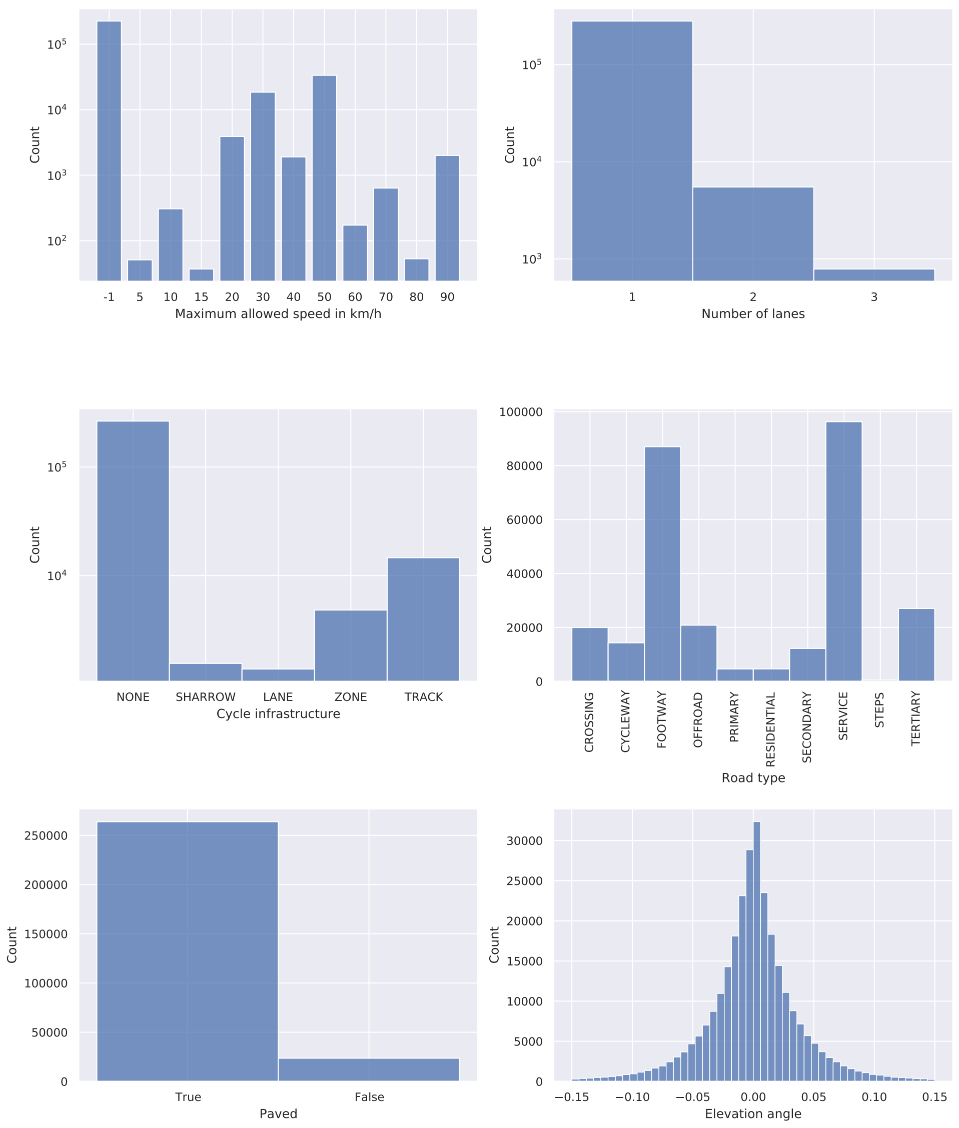


Figure 5.14: Histograms of the main features of the edges

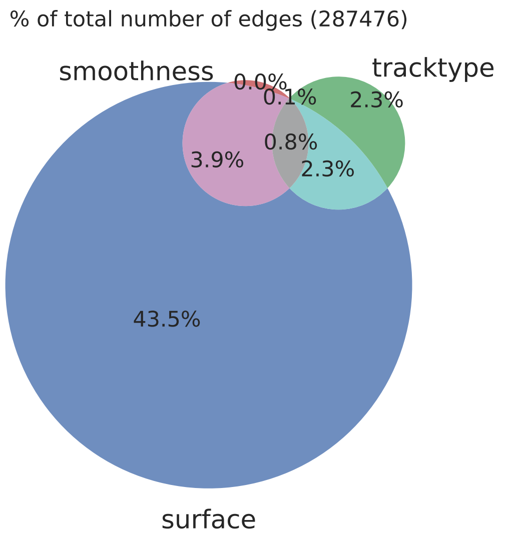


Figure 5.15: Venn diagram showing the percentage of edges with the corresponding OSM tags filled in.

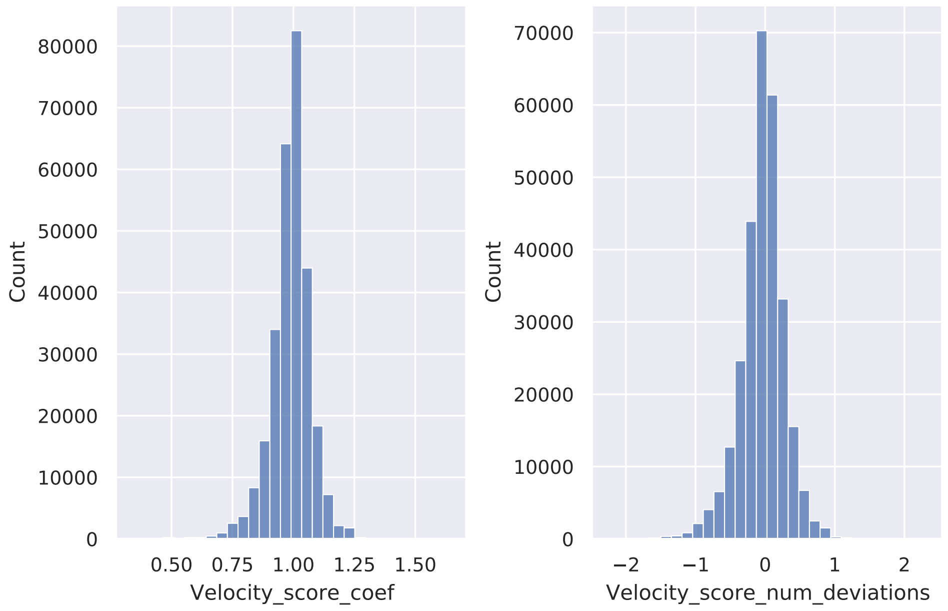


Figure 5.16: Histogram of the average user velocity scores on the graph edges

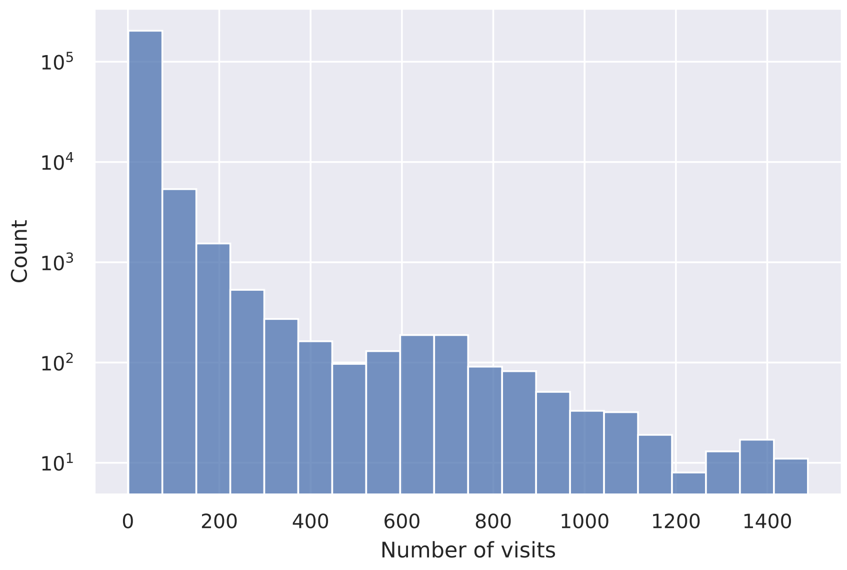


Figure 5.17: Histogram of number of visits of the graph edges

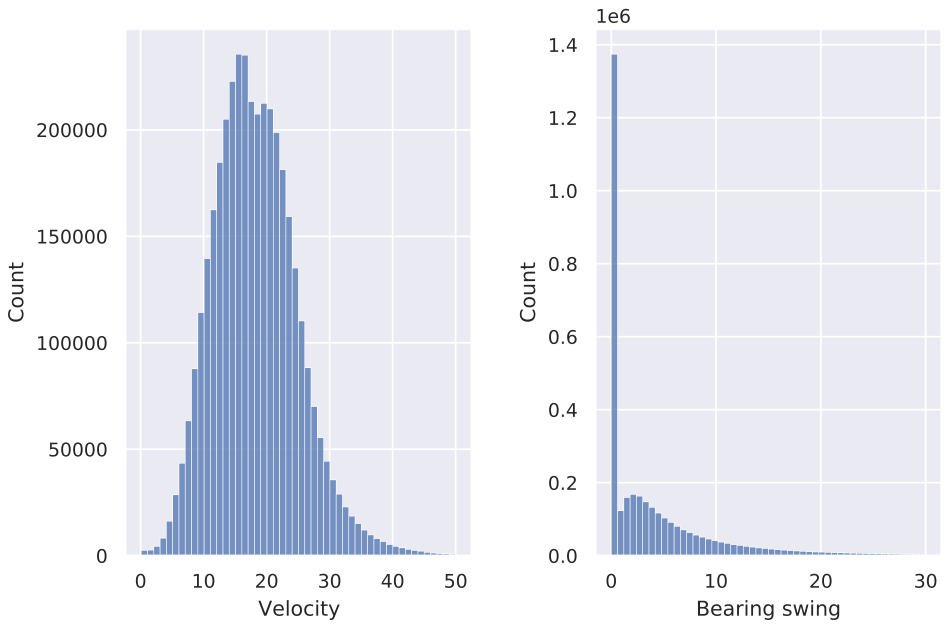


Figure 5.18: Histograms of the measurements

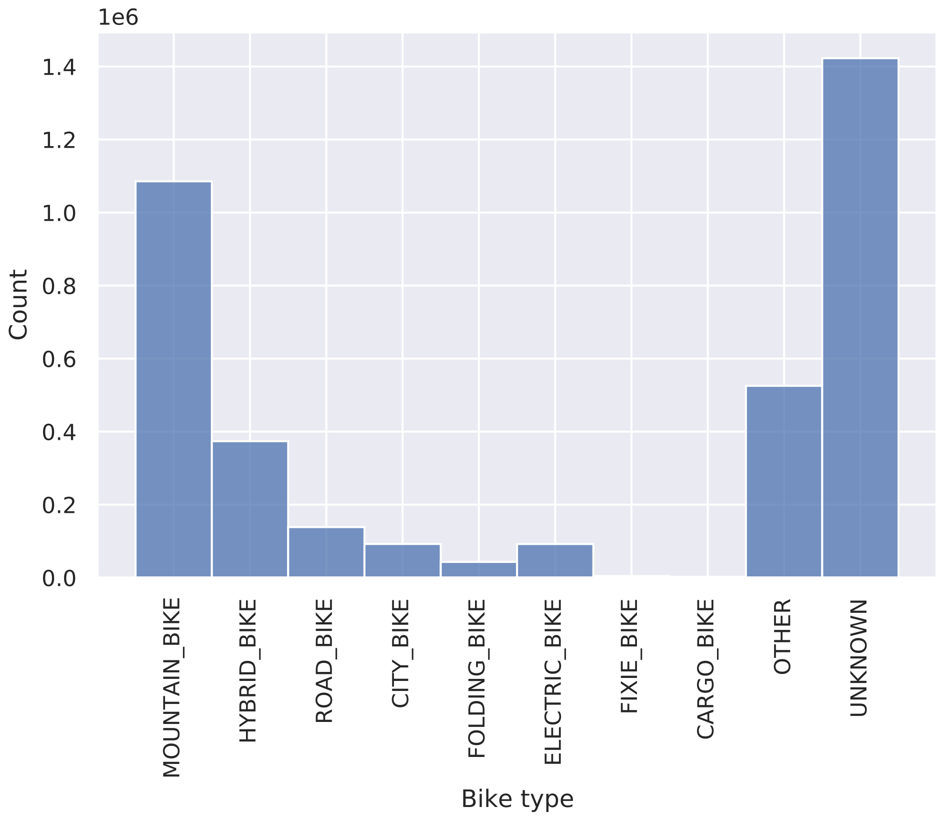


Figure 5.19: Histogram of the bike types

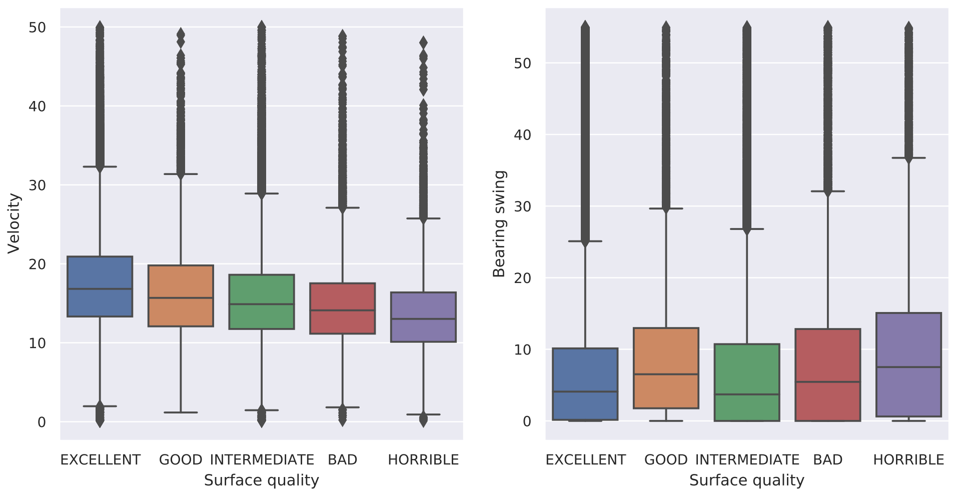


Figure 5.20: Boxplot of average travel velocities grouped by the surface quality types

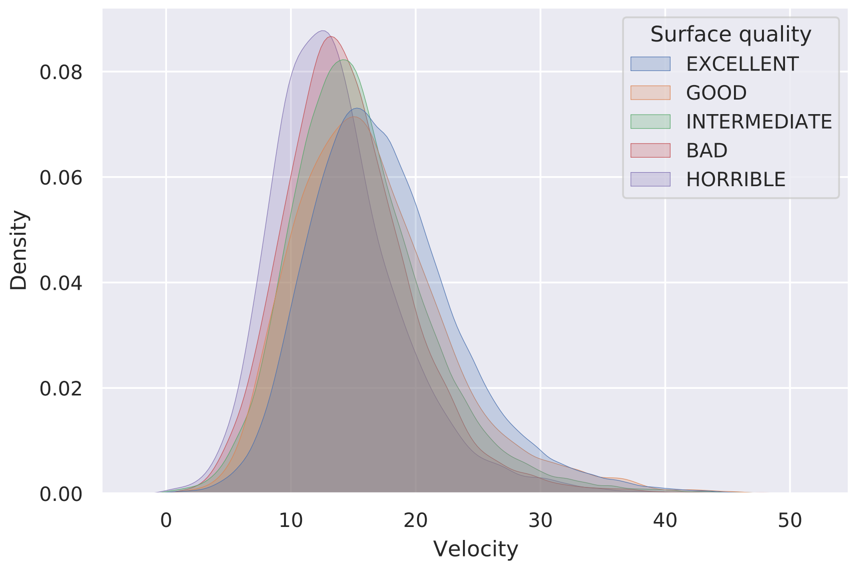


Figure 5.21: Distributions of the velocities grouped by the surface quality types. The distributions were estimated using Kernel Density Estimate method with Gaussian kernel.

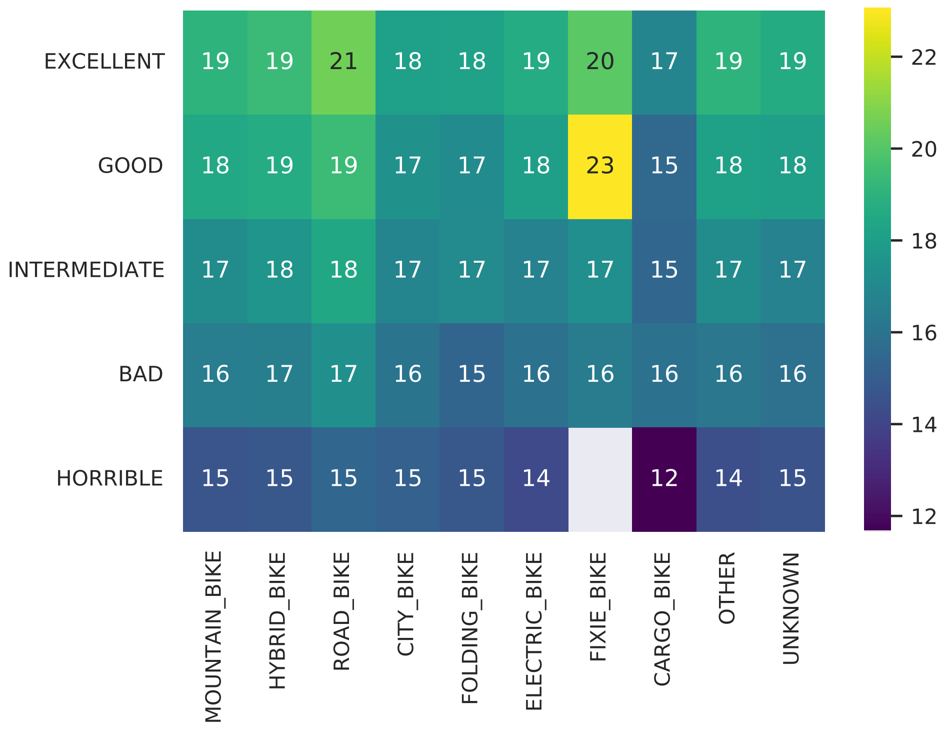


Figure 5.22: Heatmap of the average velocities on different surfaces with different bike types

Chapter 6

Solution approach

In this chapter, we will describe our approach to solving the task of the automated graph refinement by combining the machine learning techniques from Chapter 4 with the enriched navigation graph obtained using the process described in the previous chapter. First, we provide an overview of the method and describe the steps in detail. After that, we will discuss the results of the method in the following chapter.

After obtaining the enriched navigation graph, we split the edges in the navigation graph into two categories. The first category will contain edges, which we expect to have a correct surface quality already assigned. The second category will contain edges, which we suspect might have been assigned an incorrect surface quality. Then we will use data to train a model of the surface quality. The model should only use the labels of the correctly labeled edges to learn; otherwise, it could learn the mistakes in the data. However, it may use the features and connectivity of the possibly incorrectly labeled edges as well. This information is particularly useful for models, which are able to work not only with the given feature vectors but also with the underlying graph structure. This will be explained in more detail in 6.2.5. Finally, the model will be used to correct the labels of the edges in the original navigation graph.

The diagram of the whole procedure can be seen in Figure 6.1

6.1 Ground truth selection

When learning a machine learning model, the quality of the input data significantly impacts its real-world performance. The truthfulness of the labels in the training data, therefore, plays an important role in the learning process. In this thesis, we considered the correctly labeled edges the edges with at least one of the OSM tags described in 5.2.2 filled in. In the ideal case, we would only use the edges with the "smoothness" tag as it directly corresponds to the surface quality, but as we could see in Figure 5.15, these edges only form a small minority of all edges. Moreover, these edges may not be a representative sample of all edges. For this reason, we decided to consider the edges with at least one OSM tag as the ground truth, and we will try to use the model to correct the rest of the edges.

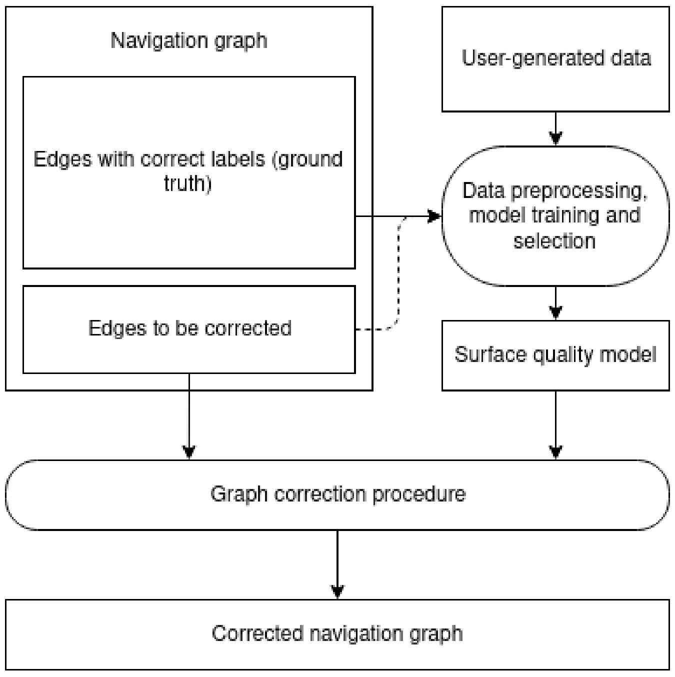

Navigation graph
Edges with correct labels (ground truth)
Edges to be corrected
User-generated data
Data preprocessing, model training and selection
Surface quality model
Graph correction procedure
Corrected navigation graph

Figure 6.1: Graph improvement procedure

We split the ground truth data into training and validation sets in the 80%/20% ratio. The training set will be used to train each model, and the validation set will be used to select the best model, which will then be used to correct the labels of the edges.

6.2 Modelling the surface quality

The prediction of the surface quality is an instance of the ordinal regression. The labels can be ordered from best quality to worst: EXCELLENT $\prec$ GOOD $\prec$ INTERMEDIATE $\prec$ BAD $\prec$ HORRIBLE. The meaning of the labels was described in 5.2.2. The labels were encoded as $k-1$ dimensional vectors according to the process we described in 4.1.3. In this section, we will describe the process of obtaining a surface quality model from the processed data. Figure 6.2 shows the overview of the process. In 7.1 we will compare the performance of the models.

6.2.1 Introduction of the used models

In this subsection we briefly describe the models we decided to use. To model the surface quality, we used four models: XGB, GNN_{reg}, $\text{GNN}_{\text{coral}}$ and GNN_{cls}. All these models are based on the models described in Chapter 4.

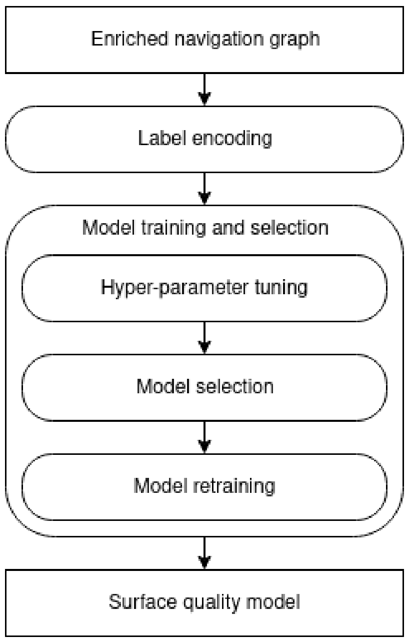


Figure 6.2: Procedure of obtaining the surface quality model

XGB

XGB is a chain of $k-1$ gradient boosted trees classifiers, each predicting one element of the encoded labels. The classifiers are chained together, which means that each classifier can use the predictions of the classifiers, which are placed before it in the chain. We think that this makes the predictions more likely to be consistent in contrast with using $k-1$ independent classifiers. For example when $Pr(y > i|x) = 0$, then it makes no sense to predict $Pr(y > i+1|x) > 0$. When the classifier knows the previous prediction, it can make use of this property. We used this model as a baseline for the graph models because even though it does not utilize the underlying graph structure, it provides a very good out-of-the-box performance on tabular data.

GNN and variants

The rest of the models are graph neural networks. The main difference among the models is in the last layer. The output layer of GNN_{reg} is a linear layer with $k-1$ outputs and the sigmoid activation. The output of $\text{GNN}_{\text{coral}}$ is the Coral layer with shared weights introduced in [14]. The output of GNN_{cls} is the linear layer with k outputs and the softmax activation. This model ignores the ordering of the labels and solves the task as classification. We included this model to verify that modifying the models to solve ordinal regression provides better results. The exact architecture of the networks will be discussed in 6.2.5. The decision to use the graph neural networks for this task was based on the fact that the connected edges, more often than not, have similar properties.

6.2.2 Model training and selection

Obtaining the surface quality model was done in the following steps:

1. Tune hyper-parameters of the models from 6.2.1
2. Select the best model based on the performance on the validation set
3. Re-train the model on all the data

All k - 1 gradient boosted trees classifiers that form the chain in the model XGB share the same hyper-parameters. Moreover, because the GNN models are very similar and share the same set of hyper-parameters, we decided to only tune the hyper-parameters for the model GNN_{reg} and use the same parameters for the other three GNN models.

6.2.3 Training implementation and details

The XGB model was trained by using the class `ClassifierChain` from the scikit-learn library [41]. This class creates the chain of classifiers, which we described in the subsection 6.2.1 using any compatible classifier. For the classifier, we used the implementation of the gradient boosted trees from the xgboost library [16]. The GNN models were implemented using the pytorch-geometric library [20] and trained using the Adam optimizer [33] for a selected number of epochs.

To compensate for the imbalance of the edge labels, we introduced sample weights. Each sample was given a weight inversely proportional to the number of samples with the same labels. When computing the average loss, it is computed as the weighted mean of the individual sample losses. This forces the model to learn to recognize all labels equally in contrast with preferring the more frequently represented labels.

Due to the memory constraints, we trained the models for each of the three locations[1] separately.

6.2.4 Hyper-parameter tuning

All used models have a wide variety of hyper-parameters. To achieve a good performance, it is essential to fine-tune these parameters. However, optimizing the performance of the model as a function of its hyper-parameters is an instance of black-box optimization, i.e., we can observe the output given the input values but generally do not precisely know how the function works on the inside. One of the popular approaches is to generate the configurations randomly from the predefined distributions and select the best-performing configuration. In this thesis, we used the Sequential Model-based Global Optimization (SMBO) approach. It works by creating a model of the optimized function, which is not costly to optimize. In each iteration, it finds the optimum of the function's model and obtains the true value by

[1]Prague and surroundings, the Jizera Mountains and the Beskid Mountains

evaluating the optimized function. Then it updates the model with the true value. We used a variant of the algorithm which models the function using the Tree-structured Parzen Estimator Approach (TPE) [8]. The input of the method is the configuration space, where each hyper-parameter is specified by a random distribution, from which we expect the optimal value to be drawn (the prior). We will not go into details about how this method works in this thesis.

Another thing that has to be decided is how to define the performance of the model. There are two main approaches:

- Performance on the validation set – set aside a portion of the training data, which will not be used to train the model. Then measure the model's performance on this validation set.
- Cross-validation – divide the training data into K parts. Then train the model K times, and in each iteration i, set i-th part aside and use it to evaluate the model's performance. The output is a list of K scores. Usually, the best model is considered to be the one with the best mean cross-validation score. The visualization of the splits can be seen in Figure 6.3.

When it is computationally feasible, it is usually better to use the second approach because it does not waste any data in the form of the validation set, and it is a better way to measure the model's performance on unseen data.

We used the 5-fold stratified cross-validation to select the best hyper-parameters for each model. This version of the cross-validation tries to split the data so that each class label is distributed evenly in each split. This is particularly useful for imbalanced datasets, which was our case. The budget for the optimization was set to 50 trials. One round of training of took around 2 minutes for the XGB model and around 10 minutes for the GNN$_{\text{reg}}$ model. The whole hyper-parameter optimization process took around 50 hours ($12 \times 5 \times 50$ minutes).

6.2.5 Model configurations

In this subsection, we will describe the configurations of the models and the final values of their tuned hyper-parameters.

XGB

Table 6.1 contains the parameters of the XGB model, their prior distributions, and their final values. The n_estimators, eta, and max_depth parameters determine the number of weak models in the ensemble, the learning rate, and the max depth of each weak model. The gamma, reg_alpha, and reg_lambda are the parameters influencing regularization and can prevent over-fitting. The subsample, colsample_bytree, and colsample_bylevel are the fractions of the samples, respectively, columns for each tree. Lower values can prevent over-fitting. min_child_weight is a parameter determining when the algorithm stops trying to split a node.

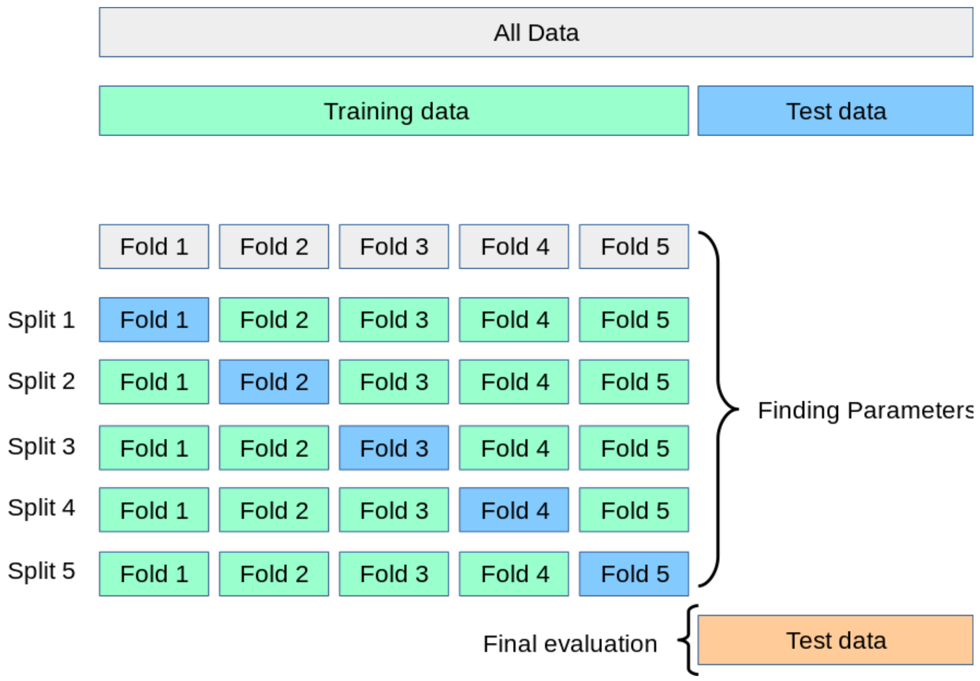


Figure 6.3: Cross-validation visualization with five splits. Image taken from [4].

	type	prior	value
n_estimators	int	uniform(100,2000)	1900
eta	float	loguniform(log(0.0001),log(0.5))	0.01
max_depth	int	uniform(1,11)	10
gamma	float	uniform(1,9)	1.04
reg_alpha	float	loguniform(log(0.0001),log(1))	0.0006
reg_lambda	float	loguniform(log(1),log(4))	2.37
subsample	float	uniform(0.5,1)	0.78
colsample_bytree	float	uniform(0.5,1)	0.93
colsample_bylevel	float	uniform(0.5,1)	0.99
min_child_weight	float	uniform(0,10)	3.36

Table 6.1: Table of hyper-parameters of XGB model

GNN

The architecture of the graph neural network models was inspired by [57]. It is summarized in Figure 6.4. It consists of several preprocess layers, several graph convolution layers, and several postprocess layers. Preprocess and postprocess layers are fully-connected, and the convolution layers consist of graph convolution operators from 6.2.5. The input of the network is the edge-based representation of the subgraph of the navigation graph formed by edges that are part of at least one trip. The preprocess and postprocess only use the features, and the convolution layers use features and the connectivity among the nodes.

	type	prior	value
num_epochs	int	uniform(200, 2000)	1800
learning_rate	float	loguniform(log(0.0001),log(0.1))	0.007
weight_decay	float	loguniform(log(0.0001),log(0.1))	0.0004
hidden_channels	int	uniform(20, 100)	90
pre_layers	int	uniform(1, 5)	2
message_passing_layers	int	uniform(1, 5)	5
message_passing_layer	conv operator	choice({GraphConv,GAT,GCN,GIN})	GraphConv
post_layers	int	uniform(1, 5)	5
dropout	float	uniform(0, 0.5)	0.0
batch_norm	bool	choice({True,False})	True

Table 6.2: Table of hyper-parameters of the GNN models

It is important to note that when we are computing the network's output, we have to evaluate the output for all the nodes in the graph. When a model works with independent samples, it is possible to evaluate it only on some part of the data. This is generally not possible to do with the graph models unless we change the structure of the underlying graph. This is the distinction between the so-called inductive and transductive learning. Transduction is reasoning from observed, specific (training) cases to specific (test) cases. In contrast, induction is reasoning from observed training cases to general rules, which are then applied to the test cases. [52] Because we want to label the specific nodes rather than create a general model, the transductive setting suits our purpose better. When training a graph model, in each epoch, we compute the output using all the nodes, but to compute the loss and the corresponding gradients, we only use the data specified for the training. This is what we meant by saying that the graph models use some information from all the data at the beginning of this chapter.

We can see that the networks have many hyper-parameters, such as the number of layers in each of the three parts of the network, the number of hidden channels, or the type of the convolution operator. All the hyper-parameters, their prior distributions, and their final values can be seen in Table 6.2. Apart from the parameters characterizing the network's architecture, there are three learning parameters: num_epochs - the number of learning epochs, learning_rate - the learning rate of the Adam optimizer, and weight_decay - regularization parameter of the Adam optimizer. We can see that the best performance was achieved without utilizing dropout. Even though dropout often prevents over-fitting and leads to better results, in the context of the graph convolutional networks, this result is consistent with the published results in [57].

6.3 Graph refinement

The best performing model obtained from steps described in 6.2.2 will be used to correct the surface quality labels of the edges we specified in 6.1. The edges will be marked with the predicted labels instead of the original ones. By doing so, we attempt to move the original navigation graph G

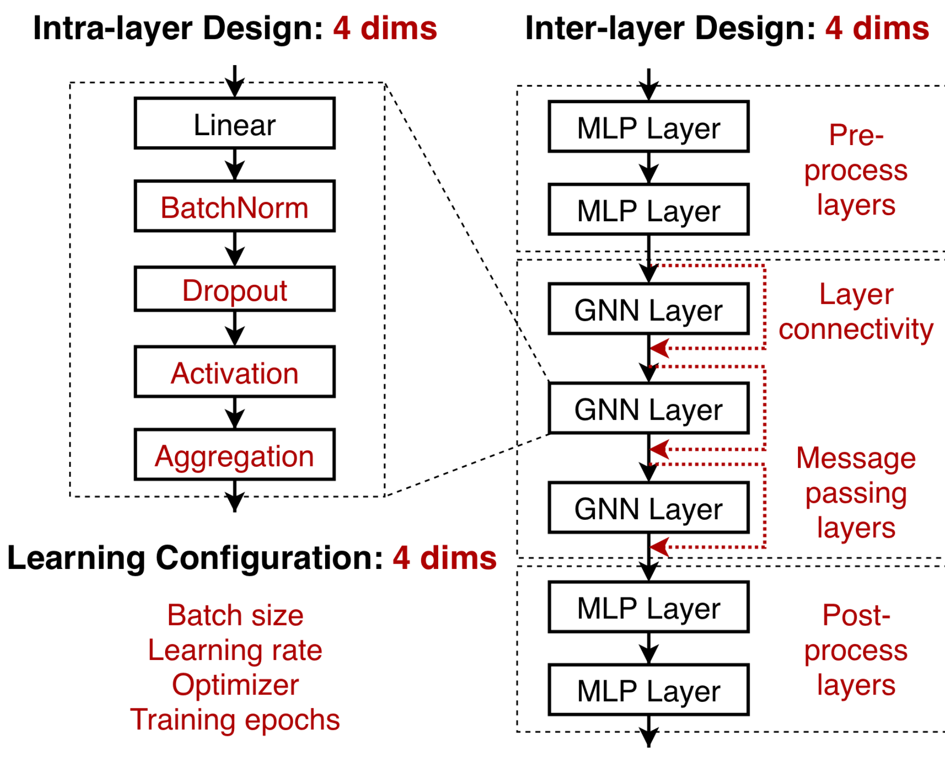


Figure 6.4: Architecture of the GNNs. Image taken from [57]

closer to the "ideal" graph G^*, therefore reaching the goal of this thesis, which we specified in Section 3.2 at the beginning of the thesis. Since we do not consider the original labels as the ground truth, we cannot measure the model's performance on the edges. However, we will manually check a small subset of the edges covered by Google Street View to get an idea of how well the model predicts the labels. Moreover, the performance of the model on the validation set provides some approximation of the model's performance on unseen data.

6.4 Implementation

This project was written Python [48] 3.10. Python is a very popular interpreted language and a very good choice particularly for machine learning projects due to the vast number of powerful libraries, that are available for free. We will list the used libraries in the order they were used in the process from data loading to results visualization.

- gpxpy [1] – parsing the GPS data in the GPX format
- pandas [51] – manipulation with the data in the tabular form
- xgboost [16] – fast implementation of the gradient boosted trees

- pytorch_geometric [20] – implementation of the graph neural networks based on pytorch [40]
- coral-pytorch [14] – implementation of the Coral layer in the $\text{GNN}_{\text{coral}}$ model
- scikit-learn [41] – a machine learning library, we used mainly its utility functions such as K-fold cross-validation
- hyperopt [9] – hyper-parameter optimization using the Tree of Parzen Estimators algorithm
- seaborn [50] – a visualization library based on matplotlib [31]
- shap [36] – a library for explaining models using the Shapely values
- kepler.gl [28] – a visualization tool for geospatial data

The experiments were run on a personal computer with an Intel i5-4590 CPU and an Nvidia GTX 970 graphical card.

Chapter 7

Results

In this chapter, we will present the results of the experiments run according to the approach we described in the previous chapter. The first section will present the performance of the four models we introduced in 6.2.1 on the ground truth part of the dataset. The second section will present the results of using the best-performing model to correct the labels of the rest of the edges.

7.1 Surface quality prediction

In this section, we will analyze the prediction error of the models and provide an explanation of the best-performing model using the feature importances.

7.1.1 Prediction error evaluation

Table 7.1 shows the performance of the models on the validation dataset in each location. We provide the regression metric RMSE and classification metric F_1-macro for each dataset separately and also the overall performance in the "All" column. The RMSE metric was computed by mapping the quality labels to consecutive integers[1]. We can see that the GNN$_{ord}$ achieved the best overall performance both in regression and classification metrics. On all datasets, the GNN models significantly outperformed the XGB model. This confirms our hypothesis that in this application, the information about the connectivity among the edges is very useful in predicting their properties. We can also see, that all models performed much better on the datasets from the mountains than in the Prague dataset. This may be due to the fact that, especially in the city center, there are factors that affect the nature of the bike traveling that are not discoverable from the data, such as traffic. We are not sure why the GNN$_{ord}$ outperformed GNN even in terms of the classification metric. GNN$_{coral}$ shows the important difference between the regression and classification metrics. It achieved significantly better results in terms of RMSE than GNN. On the other hand, its ability to predict the exact label was worse, resulting in a lower F_1-macro score.

[1]EXCELLENT $\mapsto$ 0, GOOD $\mapsto$ 1,..., HORRIBLE $\mapsto$ 4

	Prague		Jizera Mountains		Beskid Mountains		All	
	RMSE	F_1-macro	RMSE	F_1-macro	RMSE	F_1-macro	RMSE	F_1-macro
XGB	1.39	0.42	0.64	0.60	0.71	0.63	1.29	0.47
GNN	1.18	0.59	0.55	0.70	0.56	**0.76**	1.06	0.63
GNN$_{ord}$	**0.80**	**0.70**	0.52	0.70	**0.54**	**0.76**	**0.72**	**0.74**
GNN$_{coral}$	0.88	0.48	**0.49**	**0.71**	0.57	0.64	0.81	0.54

Table 7.1: The performance of the models. The best result for each dataset and metric is shown in bold. For RMSE, lower is better, for F_1-macro, higher is better.

Figure 7.1 shows the normalized confusion matrices for all models, computed from all four datasets. At first glance, we can see that the confusion matrix of GNN$_{ord}$ looks the "most diagonal." We can also see that GNN$_{coral}$ was often off by one label. Both XGB and GNN often confused the EXCELLENT label for INTERMEDIATE. Figures 7.3 and 7.4 show the confusion matrices normalized over the true and predicted values respectively.

Figure 7.2 shows the histograms of absolute errors of GNN$_{ord}$ and GNN. We can see that the counts of the errors are decreasing with the size of the error for the GNN$_{ord}$ model. GNN, on the other hand, made more mistakes of size 2 than of size 1. This is not unexpected because the GNN was trained using the classification objective function and, therefore, not penalized for larger errors.

Because GNN$_{ord}$ achieved the best overall results in modeling the surface quality, we chose to use it in the next and the most important step in the process - refining the navigation graphs.

7.1.2 Feature importances

Figure 7.5 shows the feature importances of the GNN$_{ord}$ model. The importances were computed using Shapely values [45]. The concept of Shapely value comes from the cooperative game theory, and it is a way to assign a fair payoff to each participant of a coalition. It can be used to compute the feature importances by looking at the features as the participants of a coalition and the result of the coalition being the output of the model. We can see that the most important feature was Edge_Paved, but mostly because of its importance in deciding if the surface is BAD or worse. The second most important feature was the flag indicating the edge is a crossing. We are not sure why, but from our experience, the pedestrian crossings usually have EXCELLENT or GOOD surfaces, so maybe the model learned this property. The next most important features include the measured velocity, the bearing fluctuation, the user velocity score, and the elevation angle, which are all features that we expected to play a large role in the model's decisions. By computing the feature importances, we ensured two things:

1. The important features are the ones which we expected to.
2. The model uses most of the features, i.e., there is not a small number of features that dominate in the model's decision process.

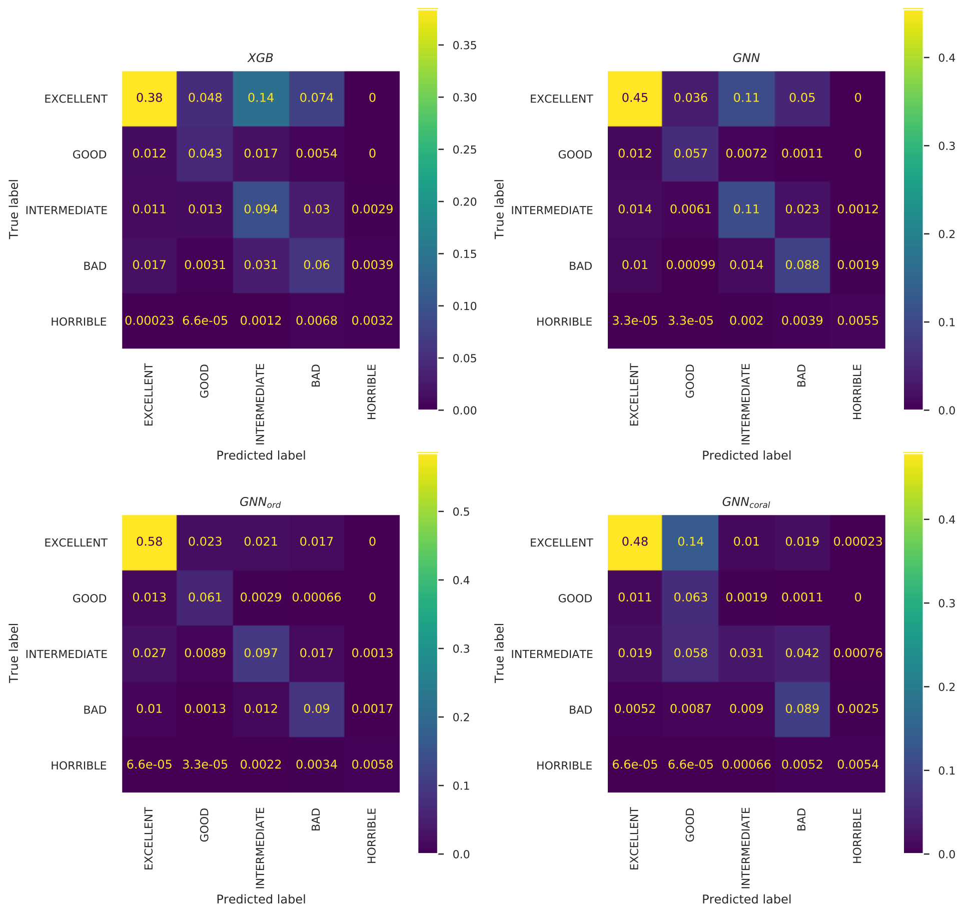


Figure 7.1: Confusion matrix of the models normalized over all values to display $P(y, \hat{y})$

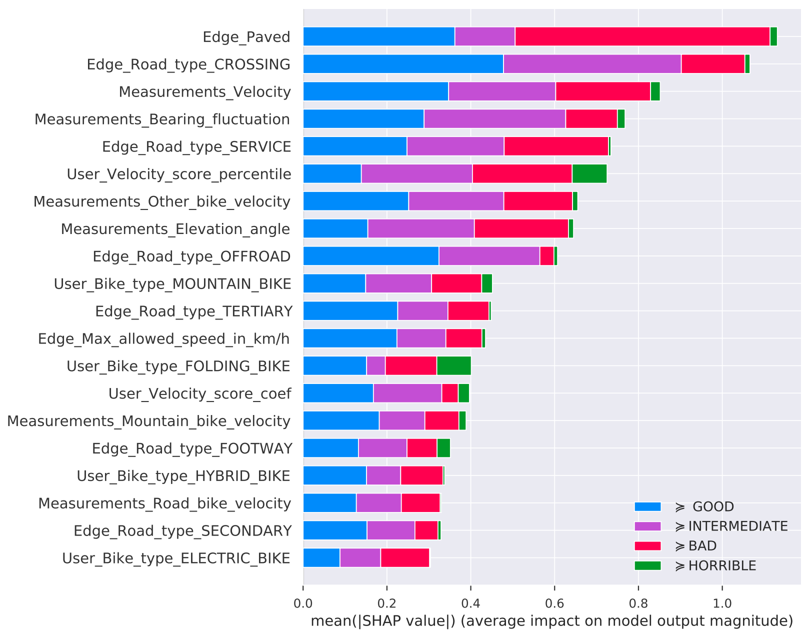


Figure 7.5: Shapely values of the features for the GNN$_{ord}$

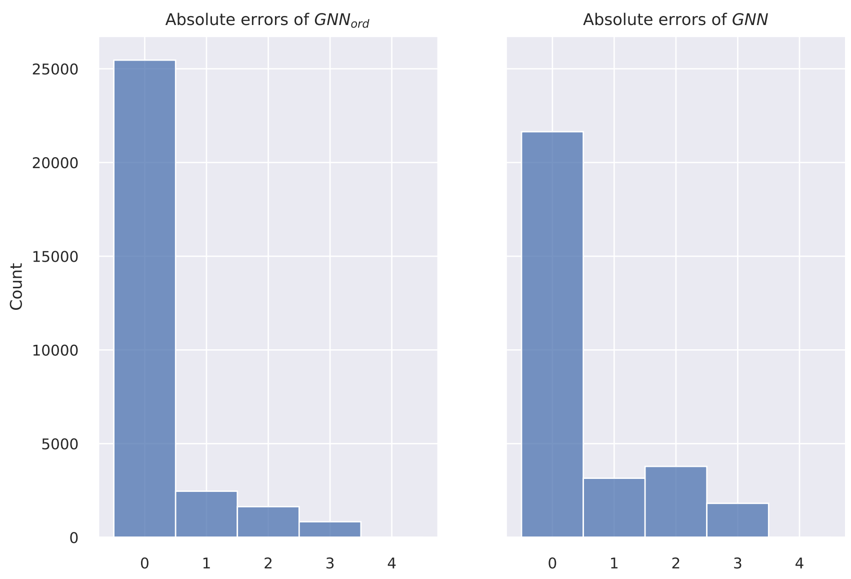


Figure 7.2: Histograms of absolute errors of GNN_{ord} and GNN.

7.2 Graph refinement

We used GNN_{ord}, the model selected in the previous section, to correct the surface quality labels of the edges with no OSM tags, as we specified in the previous chapter. In this section, we will analyze the results.

Figure 7.6 shows the confusion matrix of the model on the edges we selected to be corrected. We can see that the predicted labels correspond to the original labels much less than in the case of the validation set in the previous section. Note that this does not indicate the poor performance of the model. As we previously emphasized, the labels of these edges were derived using very simple rules, and we expected many of the original labels to be incorrect. According to our model, around 41% of the labels are incorrect, and around 30% are incorrect by two or more labels. As we saw in the previous section, our model predicted the surface quality reasonably well, but it is not perfect, so the exact numbers may be quite different.

Figure 7.7 and table 7.2 show the distributions of the original labels and corrected (predicted) labels. We can see that in Prague, there were many edges with INTERMEDIATE surface. This accounts for the fact that all pavements are automatically assigned this label. The model assigned part of the pavements to GOOD and BAD surface. We can also see that the model reassigned most of the edges with HORRIBLE surface to other labels. We do not know why the model behaved this way, but from the performance on the validation set, especially Figure 7.3, we can see that all the models had problems with correctly assigning this label, so the phenomenon may be caused by imperfections of our model.

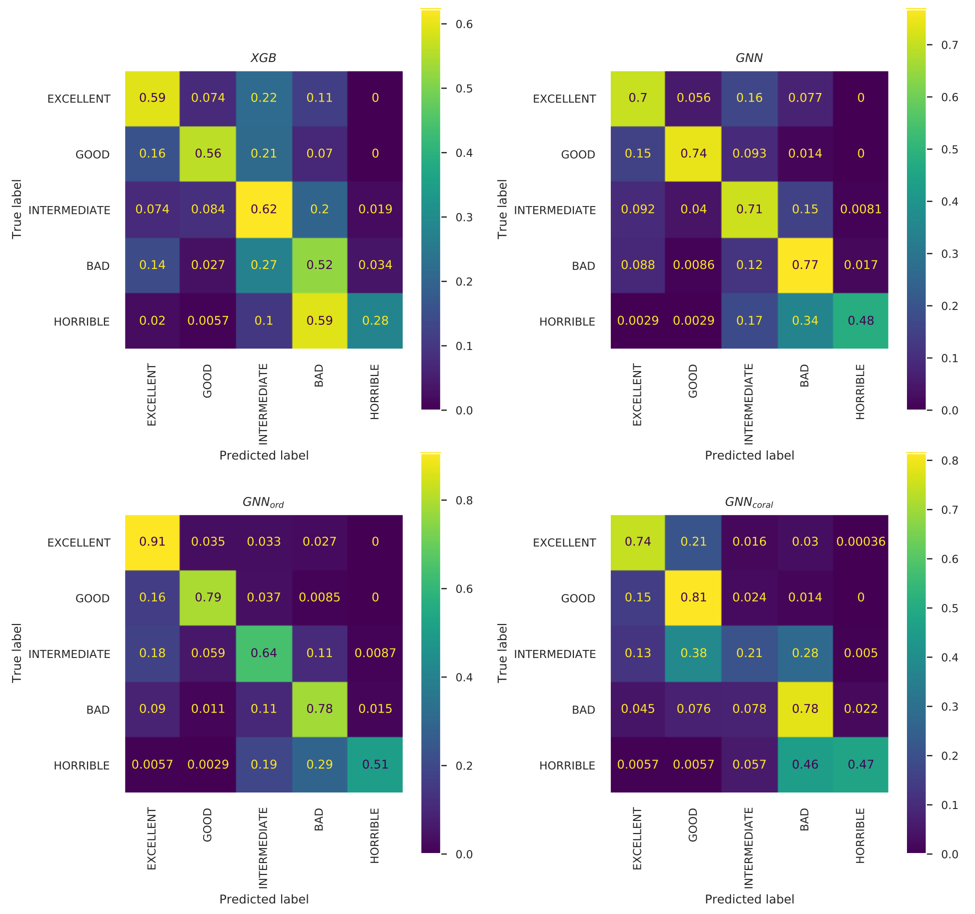


Figure 7.3: Confusion matrix of the models normalized over the true values to display $P(\hat{y}|y)$

We also analyzed the results by checking some edges that are covered by Google Street View. We will now present a few types of occurrences we found. Figure 7.8 shows an edge where the original label was EXCELLENT, and the model correctly identified it as BAD. In Figure 7.9 the original label was HORRIBLE, and the model correctly identified the edge as INTERMEDIATE. In Figure 7.10 we can see a problem that can occur in the cities. While the surface of the pavement could be considered GOOD or INTERMEDIATE, the road is BAD or HORRIBLE. The pavement and the road are two separate edges. Due to the inaccuracies of the GPS measurements, the trip may be matched to an incorrect edge. In this case, the trip was matched to the pavement, but the model predicted BAD surface. Another mistake of the model can be seen in Figure 7.11. The model incorrectly classified the road as BAD. The model may have made a mistake because the velocity on this edge was around 8 km/h, but the surface quality was not the reason the user rode so slowly. This shows the limitations of our approach in relying on the measured velocities to model the road surface.

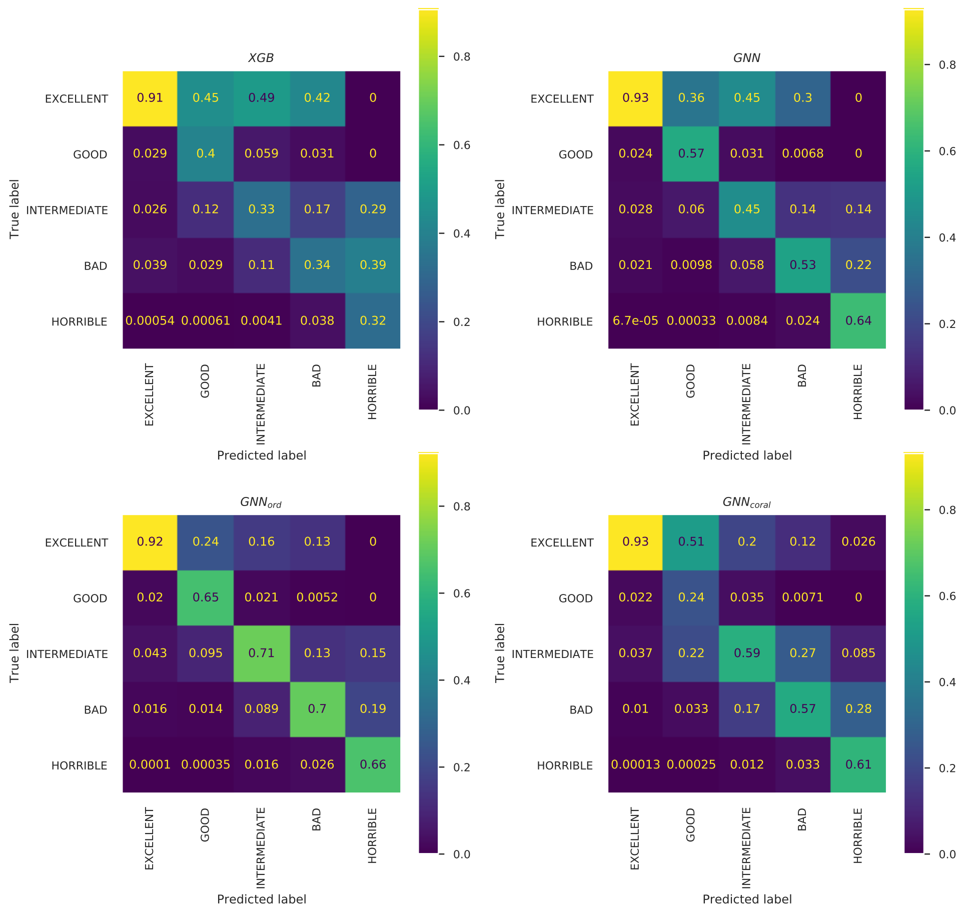


Figure 7.4: Confusion matrix of the models normalized over the predicted values to display $P(y|\hat{y})$

7.2.1 Estimate of the number of corrected edges

From the performance of the model on the validation set, we can estimate its performance in solving the task of the graph refinement. The shortcomings of this approach are:

1. The estimate of the performance on the validation set is biased because we used it to select the model. A dedicated test set would have solved this issue. However, we wanted the model to use as much data as possible.

2. We have no guarantee that the model will perform on the edges we want to correct as well as it did on the validation set because by using the edges with OSM tags for learning, we may have introduced selection bias.

However, we think that this estimate will provide a general idea of the extent to which we refined the information on the navigation graph. A very rough estimate can be done using the model's accuracy. Although we disregarded accuracy as a bad performance metric for the classification on imbalanced

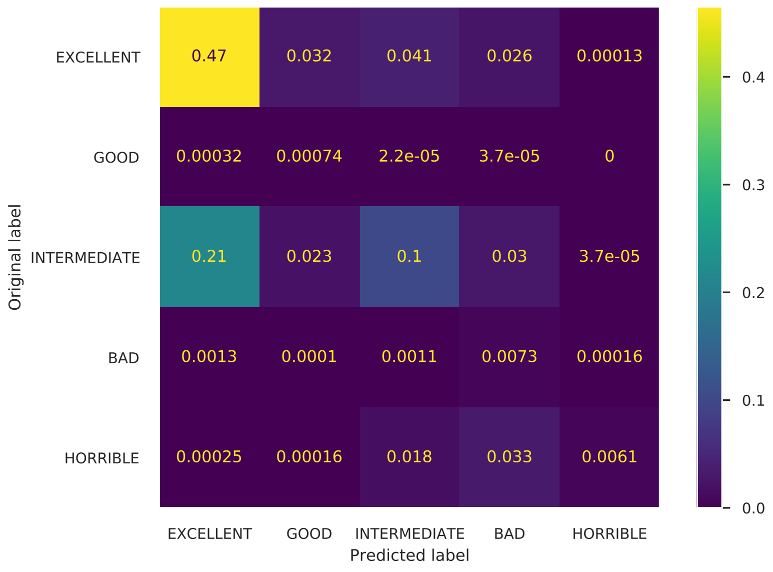


Figure 7.6: Confusion matrix on the edges to be corrected.

	Before	After
EXCELLENT	76335(0.56%)	92067(0.68%)
GOOD	151(0.0%)	7466(0.06%)
INTERMEDIATE	49880(0.37%)	22003(0.16%)
BAD	1345(0.01%)	13062(0.1%)
HORRIBLE	7761(0.06%)	874(0.01%)

Table 7.2: Distribution of the surface quality labels before and after correction.

datasets, its interpretation as the portion of correctly classified samples is useful. If we knew the label distribution, we could provide a better estimate[2], but the distribution of the labels can be quite different than in the validation dataset, and it cannot be estimated as the portion of incorrectly labeled edges can be arbitrarily large.

The error of the model on the validation dataset was 16.3%, and it predicted a different label for 41.8% of the edges. In the worst case, the model made all mistakes on already correctly labeled edges resulting in

$$(\underbrace{(41.8\% - 16.3\%)}_{\text{corrected}} - \underbrace{16.3\%}_{\text{wrongly corrected}}) = 9.2\%$$

improvement. In the best case, the model's wrong predictions were right according to the original labels. In that case, the model made a 41.8% improvement. This would mean that the model corrected between 12463

[2]by combining the prior distribution with the estimate of the conditional probabilities in Figure 7.3

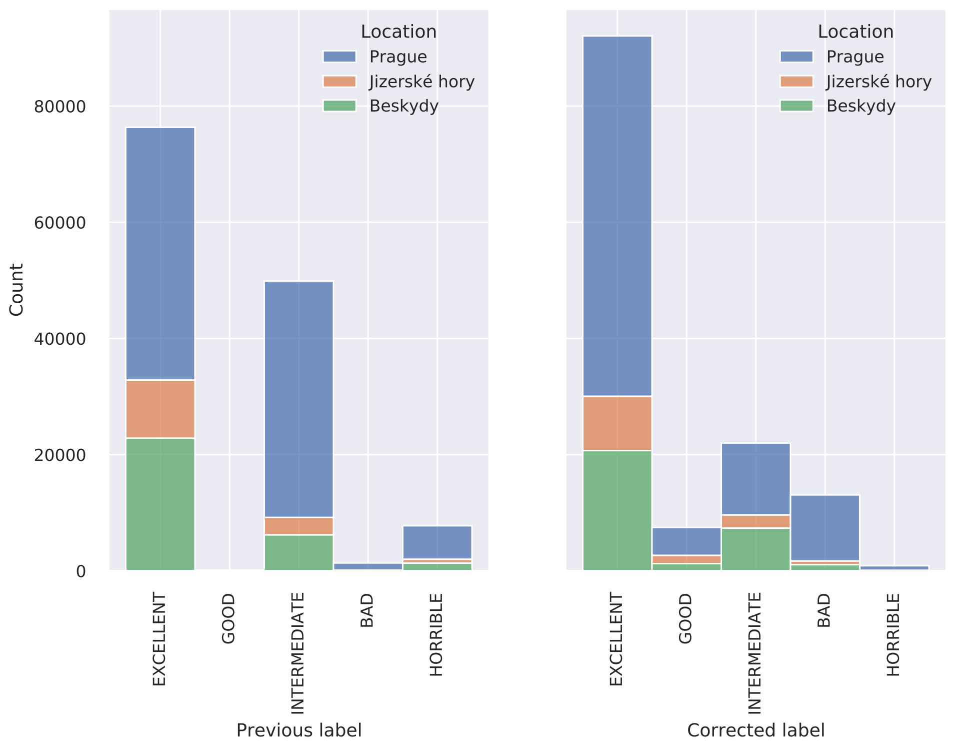


Figure 7.7: Histograms of the surface quality labels before (left) and after correction (right).

and 56627 edges. We would once again like to strongly emphasize that the distribution of the edges and the model's performance can be very different than in the validation dataset, so the true numbers can be different.

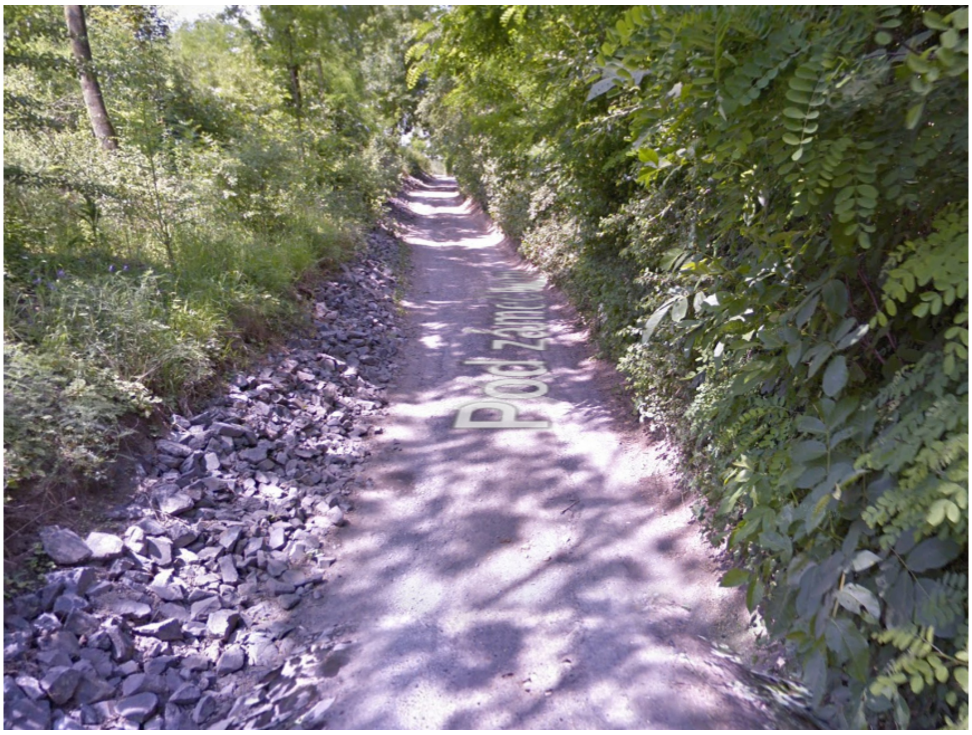


Figure 7.8: Original label: EXCELLENT, predicted label: BAD

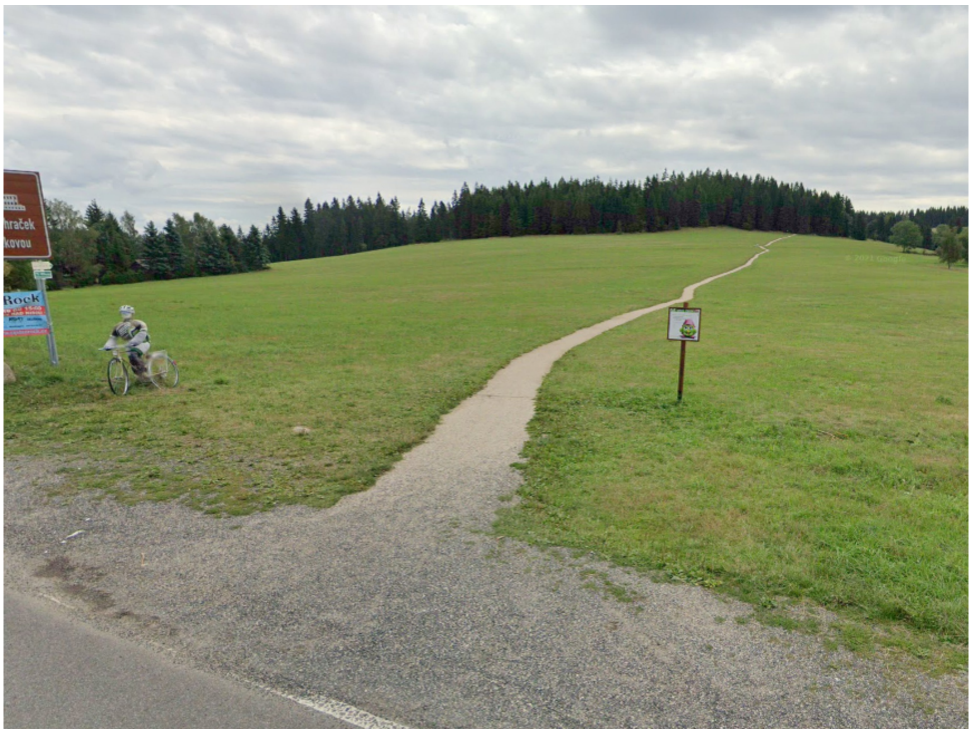


Figure 7.9: Original label: HORRIBLE, predicted label: INTERMEDIATE

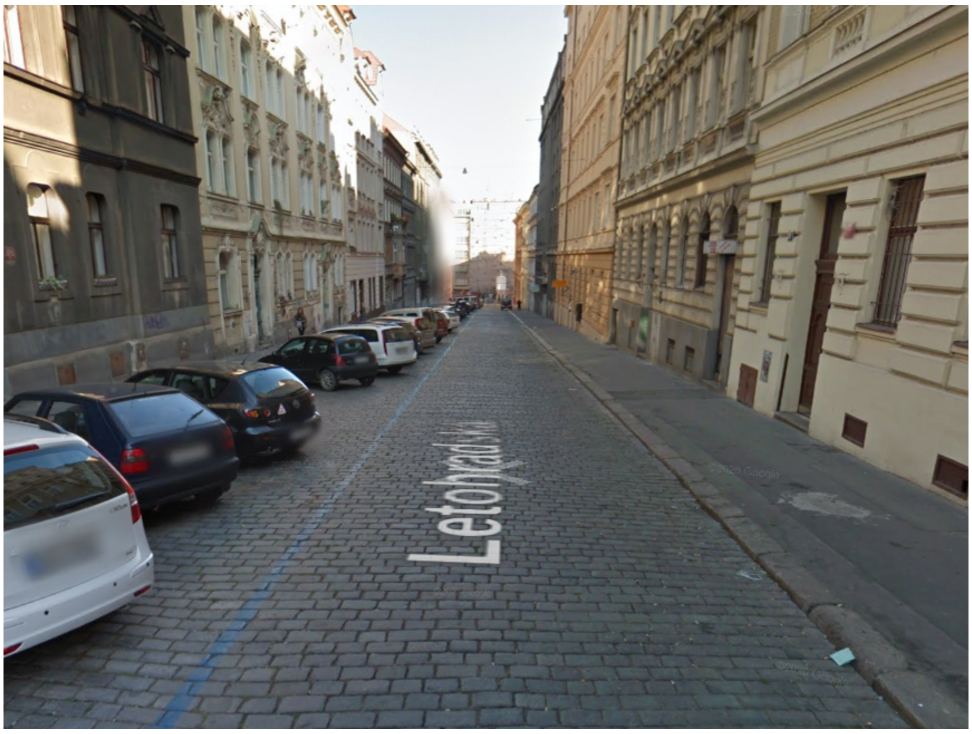


Figure 7.10: Original label: INTERMEDIATE, predicted label: BAD (edge of concern is the pavement)

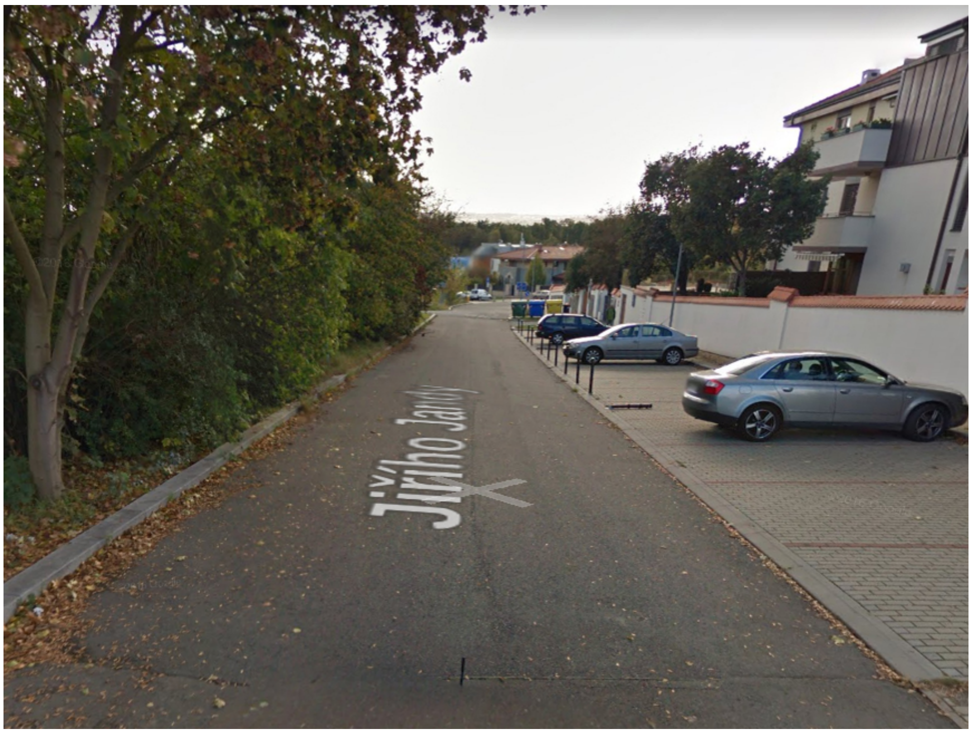


Figure 7.11: Original label: EXCELLENT, predicted label: BAD (average velocity 8 km/h)

Chapter 8
Conclusion

In this thesis, we introduced the problem of automated refinement of navigation graphs using user-generated data. First we provided an overview of the published approaches, formally defined the problem and stated the goal of the thesis, which was to refine the information about the road surface quality. Then we provided a theoretical background of the used machine learning techniques, which we used throughout the work. After that, we thoroughly described the type and contents of the data we worked with and proposed to solve the problem by modeling the surface quality using the previously described machine learning models. We proposed to consider the task of modelling the surface quality an instance of ordinal regression and to solve the task using a baseline gradient boosting model, which did not use the connectivity information in the underlying navigation graph, and graph neural networks. As a ground truth, we selected the edges with at least one OSM tag. The graph neural network models consistently outperformed the baseline model. Based on the performance on the ground truth edges, we selected the GNN$_{ord}$ model - graph convolutional network modified to solve ordinal regression, which achieved an overall performance of 0.72 RMSE and 0.74 F_1-macro. Then we used the selected model to correct the labels of the edges with no OSM tags. By making a strong assumption, that the model's performance on the edges with no OSM tags is similar to its performance on the validation dataset, we made an estimate, that the model corrected the labels of 12 to 56 thousand of edges accounting for 4.1% to 19.4% of the edges in the whole dataset.

Even though the most common approaches to model the road surface use data from high-frequency sensors such as accelerometer or gyroscope, in this thesis we have shown, that it is possible to use GPS measurements collected at large scale to achieve good results. The use of the GPS measurements is also the most obvious limitation of our approach. It is probably impossible for any model to predict the surface quality reliably with close to 100% accuracy using the data we used to train our models.

Another limitation of our approach is in the selection of ground truth. It may be the case, that only one of the OSM tags is not enough to infer the surface quality correctly. Ideally, the model should be trained using only edges, where the label was verified by the users. On the other hand, if most of the edges already had the correct label, there would be no need to solve the problem of this thesis in the first place.

Appendix A

Bibliography

[1] gpxpy – gpx file parser. https://pypi.org/project/gpxpy/.

[2] Graph neural networks. https://snap-stanford.github.io/cs224w-notes/machine-learning-with-networks/graph-neural-networks. [Online; accessed 26-April-2022].

[3] Decision tree in machine learning explained [with examples]. https://www.upgrad.com/blog/decision-tree-in-machine-learning/, Dec 2021. [Online; accessed 26-April-2022].

[4] 3.1. cross-validation: evaluating estimator performance. https://scikit-learn.org/stable/modules/cross_validation.html, 2022. [Online; accessed 26-April-2022].

[5] Touir Ameur and Rached Zantout. Design and implementation of an automatic road network map processing system using gps technology. 08 2003.

[6] Marie-Flavie Auclair-Fortier, Djemel Ziou, and Costas Armenakis. Automated correction and updating of road databases from high-resolution imagery. *Canadian Journal of Remote Sensing*, 27, 12 2002.

[7] Song Bai, Feihu Zhang, and Philip H. S. Torr. Hypergraph convolution and hypergraph attention. *CoRR*, abs/1901.08150, 2019.

[8] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[9] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 115–123, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[10] Gösta Bluhm, E Nordling, and N Berglind. Road traffic noise and annoyance - an increasing environmental health problem. *Noise & health*, 6:43–9, 07 2004.

[11] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *CoRR*, abs/2110.01889, 2021.

[12] Gabriel Cadamuro, Aggrey Muhebwa, and Jay Taneja. Assigning a grade: Accurate measurement of road quality using satellite imagery. *CoRR*, abs/1812.01699, 2018.

[13] Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *CoRR*, abs/2006.13318, 2020.

[14] Wenzhi Cao, Vahid Mirjalili, and Sebastian Raschka. Rank consistent ordinal regression for neural networks with application to age estimation. *Pattern Recognition Letters*, 140:325–331, 2020.

[15] Roberto Carisi, Eugenio Giordano, Giovanni Pau, and Mario Gerla. Enhancing in vehicle digital maps via gps crowdsourcing. In *2011 Eighth International Conference on Wireless On-Demand Network Systems and Services*, pages 27–34, 2011.

[16] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[17] Jianlin Cheng, Zheng Wang, and Gianluca Pollastri. A neural network approach to ordinal regression. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1279–1284, 2008.

[18] Rodrigo De Oliveira, Ramon Cristian Fernandes Araújo, Fabrício Barros, Adriano Segundo, Ronaldo Zampolo, Wellington Fonseca, Victor Dmitriev, and Fernando Brasil. A system based on artificial neural networks for automatic classification of hydro-generator stator windings partial discharges. *Journal of Microwaves, Optoelectronics and Electromagnetic Applications*, 16:628–645, 09 2017.

[19] Frank Ekpenyong, Allan Brimicombe, Dominic Palmer-Brown, Yang Li, and Sin Lee. Automated updating of road network databases: road segment grouping using snap-drift neural network. 01 2007.

[20] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[21] David Fiedler, Michal Cáp, Jan Nykl, Pavol Zilecký, and Martin Schaefer. Map matching algorithm for large-scale datasets. *CoRR*, abs/1910.05312, 2019.

[22] Lars Forslof and Hans Jones. Roadroid continuous road condition monitoring with smart phones. page 485–496, 2015.

[23] Vojtěch Franc. Statistical Machine Learning (BE4M33SSU) Lecture 2: Empirical Risk. `https://cw.fel.cvut.cz/b201/_media/courses/be4m33ssu/er_zs2020.pdf`, 2020.

[24] Eibe Frank and Mark Hall. A simple approach to ordinal classification. volume 2167, pages 145–156, 08 2001.

[25] Tiago R. M. Freitas, Antoanio Coelho, and Rosaldo J. F. Rossetti. Improving digital maps through gps data processing. In *2009 12th International IEEE Conference on Intelligent Transportation Systems*, pages 1–6, 2009.

[26] Tiago R. M. Freitas, António Coelho, and Rosaldo J. F. Rossetti. Correcting routing information through gps data processing. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 706–711, 2010.

[27] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[28] Shan He. From beautiful maps to actionable insights: Introducing kepler.gl, uber's open source geospatial toolbox. `https://eng.uber.com/keplergl/`, Mar 2020.

[29] Songtao He, Favyen Bastani, Satvat Jagwani, Edward Park, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Samuel Madden, and Mohammad Amin Sadeghi. Roadtagger: Robust road attribute inference with graph neural networks, 2019.

[30] Johannes Heidt and Klaus Dorer. Classification and prediction of bicycle-road-quality using imu data. *APPLICATION IN LIFE SCIENCES AND BEYOND*, page 138.

[31] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[32] Ahmad Ismail, Nur Mat Nuri, Mazlan Mansor, Muhamad Shukri, and Mohamad Afiq. Study on the road transmitted vibration of a mountain bicycle. *Jurnal Teknologi*, 77, 12 2015.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[34] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[35] Stefan Litzenberger, Troels Christensen, Otto Hofstaetter, and Anton Sabo. Prediction of road surface quality during cycling using smartphone accelerometer data. *Proceedings*, 2:217, 02 2018.

[36] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[37] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[38] Mark R Miller and David E Newby. Air pollution and cardiovascular disease: car sick. *Cardiovascular Research*, 116(2):279–294, 10 2019.

[39] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. *CoRR*, abs/1810.02244, 2018.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[41] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[42] S. Russell, S.J. Russell, P. Norvig, and E. Davis. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 2010.

[43] Michael W. Sayers. The international road roughness experiment (irre) : establishing correlation and a calibration standard for measurements. 1986.

[44] I. K. Sethi and G. P. R. Sarvarayudu. Hierarchical classifier design using mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(4):441–445, 1982.

[45] Lloyd S. Shapley. *Notes on the N-Person Game mdash; II: The Value of an N-Person Game*. RAND Corporation, Santa Monica, CA, 1951.

[46] Philip H. Swain and Hans Hauska. The decision tree classifier: Design and potential. *IEEE Transactions on Geoscience Electronics*, 15(3):142–147, 1977.

[47] Renate van der Zee. How amsterdam became the bicycle capital of the world, May 2015.

[48] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[49] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.

[50] Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[51] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[52] Wikipedia. Transduction (machine learning) — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Transduction%20(machine%20learning)&oldid=1013928983`, 2022. [Online; accessed 26-April-2022].

[53] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Le Song. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Singapore, Singapore, 2022.

[54] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[55] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018.

[56] Jungkeun Yoon, Brian Noble, and Mingyan Liu. Surface street traffic estimation. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services*, MobiSys '07, page 220–232, New York, NY, USA, 2007. Association for Computing Machinery.

[57] Jiaxuan You, Rex Ying, and Jure Leskovec. Design space for graph neural networks, 2020.

[58] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[59] Qiaoping Zhang and Isabelle Couloigner. A framework for road change detection and map updating. volume 35, 07 2004.

Appendix B

Attachment contents

/

`datasets`	a sample of the used data
`data_tools`	a python module for working with the data
`universal`	universal utility functions
`dpnk_tools`	functions for working specifically with our data
`documents`	the digital version of this thesis
`models`	implementation of the used models and their training
`notebooks`	Jupyter norebooks used mainly to produce the visualizations
`README.md`	text file with technical instructions for the project
`requirements.txt`	text file with needed dependencies