MASTER'S THESIS | LUND UNIVERSITY 2018

Improving the semantic accuracy and consistency of OpenStreetMap using machine learning techniques

Michal Stypa, Hannes Sandberg

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2018-24

Improving the semantic accuracy and consistency of OpenStreetMap using machine learning techniques

Michal Stypa
dic12mst@student.lu.se

Hannes Sandberg
dic12hsa@student.lu.se

January 7, 2018

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Pierre Nugues, Pierre.Nugues@cs.lth.se

Examiner: Elin Anna Topp, Elin_Anna.Topp@cs.lth.se

Abstract

OpenStreetMap (OSM) is a collaborative project with the aim of creating a free editable map of the world. It is considered one of the most prominent examples of Volunteered Geographic Information with the majority of data generated and edited by user contributors. The OSM dataset describes real-world phenomena by associating a set of attributes to geographic primitives. The semantics of each entity are described using structured key/value pairs called *tags*. Due to their simple and open semantic structure, the approach often results in noisy, inconsistent, and ambiguous data.

In this thesis, we explore possible methods of improving the quality of OpenStreetMap data in terms of attribute accuracy and consistency using machine learning techniques. During the process, we identify major challenges together with viable solutions in an end-to-end application. The resulting system is capable of highlighting aberrations and predicting true values for every attribute with at least 80% accuracy depending on the attribute.

Keywords: openstreetmap, machine learning, data mining, feature engineering, decision trees

Acknowledgements

We would like to express our sincere gratitude to Daniel Palmqvist at Apple for devoting his time to this project. With his knowledge, commitment and open mind, he gave us the opportunity to explore new problems and develop as engineers. We would also like to thank Pierre Nugues for his invaluable advice and compassionate guidance. Finally we would like to thank Stefan Christiansen at ÅF, the entire ÅF IT Team and all Apple employees affiliated to this thesis for their warm and kind reception.

Contents

Chapter 1
Introduction

1.1 Purpose

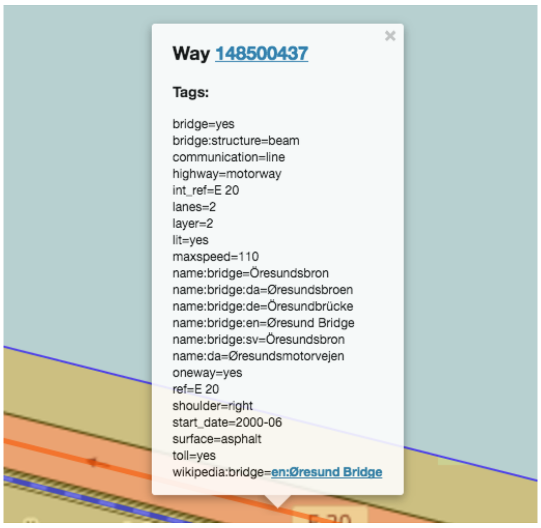
The objective of this thesis is to explore possible methods of improving the quality of OpenStreetMap data in terms of attribute accuracy and consistency using machine learning techniques. Such improvements are necessary to obtain a consistent description practise reducing data noise, increasing the amount of information associated with each object and giving a more accurate description of the reality. The main focus of our work is to identify challenges together with solutions in an end-to-end application providing both error prediction and support while creating new or editing existing data.

1.2 OpenStreetMap

OpenStreetMap is a collaborative project with the aim of creating a free editable map of the world. It is considered one of the most prominent examples of Volunteered Geographic Information (VGI) (Wikipedia, 2017), where the majority of data is generated and edited by common users. OSM has proven to be accurate and extensive enough to be considered a serious alternative to commercial map services like Google and Apple Maps. Despite constant improvements and a fast growing number of contributors (OpenStreetMap Wiki, 2017d), OSM data suffers (as many other Open Source projects of this kind) a quality deficit in less populated regions, especially in terms of coverage and feature extent.

Positional accuracy and coverage degree can and have been measured by comparison with commercial map services and datasets provided by governmental mapping agencies. The real challenge however, lies in feature validation i.e. the semantic accuracy of each feature and the consistency with which a feature is used in different scenarios (Haklay, 2010). Figure 1.1 shows the vast differences in description consistency between two similar objects.

Furthermore, a study conducted by Ali et al. (2014) addressing the problematic nature of ambiguous feature classifiers (both tag keys and values) showed that contributors often disagree among themselves over which descriptions to use given a certain element.

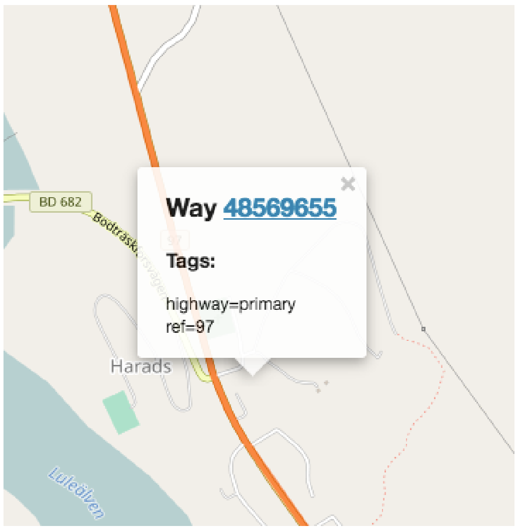


Way 148500437

Tags:

```
bridge=yes
bridge:structure=beam
communication=line
highway=motorway
int_ref=E 20
lanes=2
layer=2
lit=yes
maxspeed=110
name:bridge=Öresundsbron
name:bridge:da=Øresundsbroen
name:bridge:de=Öresundbrücke
name:bridge:en=Øresund Bridge
name:bridge:sv=Öresundsbron
name:da=Øresundsmotorvejen
oneway=yes
ref=E 20
shoulder=right
start_date=2000-06
surface=asphalt
toll=yes
wikipedia:bridge=en:Øresund Bridge
```

(a) An example of a well tagged entity - a stretch of the Öresund Bridge connecting Sweden and Denmark.



Way 48569655

Tags:

```
highway=primary
ref=97
```

(b) An example of a poorly tagged entity - a stretch of Swedish highway 97 between Luleå and Jokkmokk.

Figure 1.1: Two OSM road entities illustrating the vast difference in tagging consistency.

1.2.1 Architecture

The OSM dataset describes real-world phenomena by associating a set of attributes to geographic primitives. The attributes of each *element* (a point, area boundary, or relation) are described using structured key/value pairs called *tags* describing the semantic meaning of the particular element (OpenStreetMap Wiki, 2017a). All elements consist of at least one node – the smallest entity in the dataset. At its core, a node contains an *id* and the coordinates of the node denoted in a latitude and a longitude value. To connect one or more nodes, the OSM dataset uses elements called *ways*. A way element can be anything from a stretch of a road to a building outline or an area boundary. The way contains an ordered list of nodes defining its anchor points and position.

1.2.2 Example

To better illustrate the interaction between nodes, ways and tags, consider this example of a *way* element – a stretch of the E6 motorway in Sweden, see Figure 1.2. Typically, such an element will contain a list of nodes defining its position and shape and the tags listed in Table 1.1 describing its properties.

The description pattern described in Table 1.1 will be found in most parts (way elements) of the E6 motorway, but not all of them. Some users may want to add the *layer* = *0* tag as the road is positioned on ground level. Some may add the tag *foot* = *no* marking the

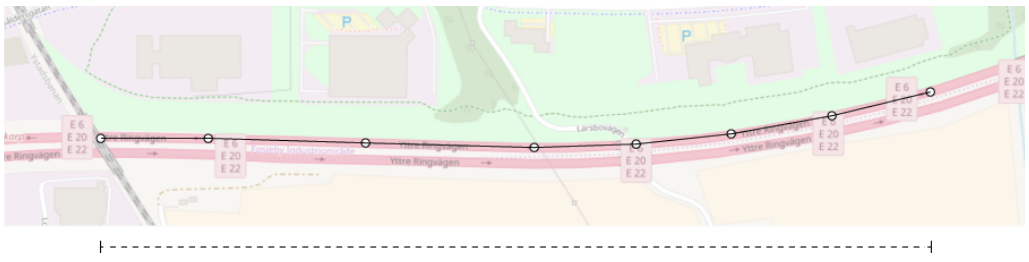


Figure 1.2: An example of an OSM entity – a stretch of the E6 motorway. Nodes denoted as circles, way denoted as the solid black line.

Table 1.1: Tags associated with the example element in Figure 1.2

Key	Value	Tag description
highway	motorway	the type of road
maxspeed	110	the speed limit expressed in km/h
ref	E06	the reference number
name	E6	the name of the road
lanes	2	the number of lanes
oneway	yes	binary value describing weather the road is a one-way road or not

road as forbidden for pedestrians. Both descriptions are correct but do not comply with the general patterns of OSM as they can be derived from implications. These examples of structural aberrations make the data noisy and inconsistent. In an analytic context, each new feature added to an element of the same kind will extend the number of possible features for all other similar elements. Furthermore, the data itself can contain errors without deviations from the general pattern. A speed limit of 20 km/h in our example element is most probably incorrect. Such an error can be discovered and highlighted/fixed by using the patterns associated with the other tags in the element where a motorway with two lanes typically has much higher limits.

1.3 Research questions

To solve the problem described in Section 1.1, the following questions are addressed and answered:

- How can defect entities be found using machine learning techniques?
- Which features are wrong in a defect entity?
- How can the proper value of a feature be derived?

1.4 Related work

There have been several projects aiming to improve the quality of OSM data. A majority of these target positional and structural accuracy leaving the semantics aside. Trajectory mining by Basiri et al. (2016) is an example of positional accuracy improvement systems utilizing GPS trails provided by anonymous users. Most error detecting systems are built using static if-statements looking for structural errors and other flaws easily detected by a set of handcrafted rules.

The attempt closest to our approach was a software developed as a result of a semantic similarity study of the OSM wiki by Vandecasteele and Devillers (2013). The OSM Semantic Network was developed by parsing the wiki using an altered version of the search engine algorithm used by Google called PageRank. The Penetration-Rank algorithm computes a network based on nodes, tags in this case, referencing and being referenced by other nodes. The software was deployed in the official OSM editing software JOSM giving the editors suggestions in a fashion similar to ours. The suggestions were based on the similarity computed by the network looking at objects within close proximity. For unknown reasons, the plugin was discontinued and is no longer available in the JOSM plugin manager. The approach was a good try to improve the semantic accuracy of OSM but had a major drawback. The system was built on wiki-page references and not the actual data. The network could only detect closely related tags by page correlation for each tag, ignoring the influence of other tags present in the element.

We did not manage to find any related work using machine learning to improve the semantic accuracy of OSM. Countless studies have of course been performed in the machine learning field where similar methods are used in different contexts.

1.5 Limitations

One of the biggest challenges in any machine learning context is the evaluation process. Obtaining reliable results and verifying their reliability requires extensive testing and sufficient amounts of data. Due to the limited time scope of this thesis, we had to make certain assumptions based on previous experiences and literature. To evaluate newly introduced features, we relied upon evaluation performed on randomly chosen fractions of the main dataset.

1.6 Contributions

The entire thesis was conducted in complete collaboration.

Chapter 2
Theory

This chapter explains the fundamentals of machine learning and the processes used in the method of this report.

2.1 Machine learning

Machine learning (ML) is a subfield of computer science, which evolved from the study of pattern recognition and statistics. By exploring patterns in input data, ML algorithms derive models used to make predictions on novel input without the need of explicit programming. Learning in this context means an agent's ability to improve on future tasks by making observations about the previously provided data. There are three different types of machine learning:

- Unsupervised learning – The agent learns patterns in the input without explicit feedback. The most common unsupervised learning task is clustering, where input data is clustered by similarity (such as minimum Euclidean distances between numerical values).
- Supervised learning – The agent derives a mapping function between input and output values supplied in a training set.
- Reinforcement learning – The agent is either rewarded or penalized based on its performance given a certain task. The retroactive feedback is used to improve the agent model.

This thesis focuses on supervised learning as both input and output values are provided in training datasets. Unsupervised learning was used in an imputation attempt but was later abandoned, see Section 3.4.3.

Regardless of the machine learning algorithm, the supplied data needs to be transformed into a N-dimensional matrix where each entity (observation) corresponds to one row in the matrix $\mathbf{x_i} = \{ x_1, x_2, ..x_n\}$. This is of significant importance when dealing with OSM data as each new attribute extends the matrix by one N-dimensional column for every instance present in the dataset (Russell and Norvig, 2010).

Table 2.1: Input vector matrix

$\mathbf{x_i}$	x_1	x_2	...	x_n
$\mathbf{x_1}$	$value_1$	$value_2$	...	$value_n$
$\mathbf{x_2}$	$value_1$	$value_2$	...	$value_n$
...				

2.1.1 Classification and regression

A training set in supervised learning consists of N example input-output pairs.

$$(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), ...(\mathbf{x_N}, y_N)$$

As each y_j is generated by an unknown function $y = f(\mathbf{x})$, the goal is to discover a function h that approximates the true function f. The function h is called the *hypothesis* and aims to generalize f to correctly predict the value of y for novel instances of x. When the output y is a finite set of values, the problem is called *classification*. When y is a number, the problem is called *regression* where the predicted output is a conditional expectation or an average value of y as the probability of finding the exact value of y is 0. (Russell and Norvig, 2010)

2.1.2 Decision Trees – J48

A decision tree represents a function that outputs a single decision given a vector of attribute values. Each tree node A_i corresponds to a test of the values of one of the input attributes with branches labeled with all possible values of the assessed attribute. A decision is made by reaching one of the leaf nodes where the return value is specified, see Figure 2.1. The main advantage of decision trees over other classification algorithms such as *support vector machine, naive bayes and neural networks* is the natural representation of the classifier model. The tree is easily understood by humans where all decisions can be traced by simply inspecting the tree.

J48 is a Java implementation of the C4.5 algorithm used to generate decision trees. C4.5 constructs the tree in a top-down manner by exploiting subset impurity called *entropy*.

$$Entropy : H(X) = -\sum_k p(x_i) \log_2 p(x_i) \tag{2.1}$$

Splitting the main dataset on each attribute and comparing the entropy before and after the split yields the *information gain* for each attribute. The attribute with highest information gain is chosen as the root node while attributes with lower gain are placed as children in descending order. Overly complex trees may suffer from *overfitting* as the number of

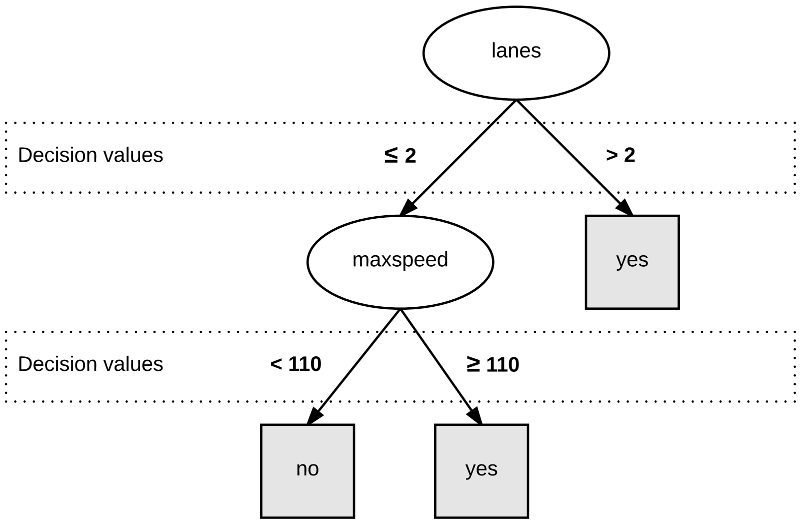


Figure 2.1: A simplified decision tree determining whether the input $x = \{lanes, maxspeed\}$ is a motorway.

attributes and/or their values relative to the number of observations grows. Overfit trees fail to capture general trends and perform poorly on novel instances. To combat this issue, the tree needs to be *pruned*. C4.5 uses both statistical confidence estimates and a minimal number of instances within each node to decide whether to keep a node or make it a leaf and rely on the nodes further up the tree. The error confidence interval p is calculated as follows.

$$p = f \pm z\sqrt{\frac{f(f-1)}{N}} \tag{2.2}$$

Where f is the observed error rate measured over the set of N training instances and z is a factor dependent on the desired level of confidence C. The upper bound of the produced error estimate is used for pruning as it makes for a pessimistic assumption for future error rates (Witten et al., 2016). In this thesis, we utilize decision tree algorithms to construct classification models in order to predict the most probable tag values given a certain element.

2.1.3 Association analysis – Apriori

The Apriori algorithm is used to mine frequent itemsets for Boolean association rules highlighting general trends in datasets. Itemsets are considered frequent when the occurrence of the itemset is higher then a pre-defined threshold *min_support*.

$$supp(X) = P(X) \tag{2.3}$$

The algorithm uses a level-wise search where k-itemsets are used to explore (k+1)-itemsets thus extending the tested itemsets by one item in each iteration. The process, known as *candidate generation*, utilizes the fact that any subset of a frequent itemset must also be a frequent itemset thus reducing the computation cost. The generated candidate list is used to derive association rules $X \Rightarrow Y$ satisfying the minimum *confidence* (Agrawal et al., 1994). In this thesis, we use unsupervised association rule mining in an attempt to impute missing data.

$$conf(X \Rightarrow Y) = \frac{supp(X \Rightarrow Y)}{supp(X)} \tag{2.4}$$

2.1.4 Missing values and imputation

In statistics, an absence or malformation of a data point in an input vector instance is commonly referred to as a *missing value*. To minimize information loss caused by incomplete input data, missing values can be *imputed*. There are several different imputation techniques but no *'best practise'* as the suitability of each method highly depends on the nature of used datasets and their features (Musil et al., 2002). Differences in tagging consistency i.e. the frequency and consequence with which a feature is used in different elements result in a lot of missing values. In this thesis, we impute missing values in OSM entities using both common techniques and machine learning.

Mean substitution	Impute the value using the mean of values available in complete instances. Works only for numeric values.
Hot-deck	Sort the dataset and impute the value from a neighbouring instance.
Random draw	Impute by selecting a random value among other complete instances.
Most common	Impute with the most common value for the given feature.
Regression	Predict the value using regression based on complete instances.

2.2 Accuracy evaluation

2.2.1 Training, test, and validation

For evaluation purposes, the main dataset is divided into a *training set* and a *test set*. The classifier is provided with the training set as input data. The evaluation is done by running the test set and comparing the predicted output with actual values.

2.2.2 Percentage split

Percentage split randomly selects $S_{training}\%$ of the data for training and $100 - S_{training}\%$ for testing. Observations used for testing are therefore omitted during the training phase thus making the technique unsuitable for small datasets.

2.2.3 Cross-validation

Cross-validation can be used to minimize information loss during the training and evaluation phase. *k-fold* cross-validation partitions the data into k folds. Each fold is used as a test set while all other folds serve as training data, repeated *k* times. The main disadvantage of cross-validation is the resource consuming process of training the model *k* times.

2.2.4 Validation

When performing extensive training and testing, too much specific information about the test set might be included in the model causing an overfit. To ensure that the model is general enough to handle unseen data, a third partition of the dataset can be used for validation.

2.2.5 F-measure

A simple performance measure for any classification model is the accuracy e.g. the number of correctly classified instances compared to the total number of instances. Although this method provides a viable value, it is highly dependent on the test data and does not reveal the overall performance of the model. An accuracy of 98% in a test set where 97% of all instances yield the same output is not a valid measure of the models performance as the model may be skewed towards favouring the output present in the majority of the data. To obtain a complete evaluation of a models performance, following statistical measures must be accounted for, see table 2.2.

Table 2.2: Measures

Measure	**Description**
True positives (TP)	Instances classified as *y* with actual value *y*
False positives (FP)	Instances classified as *y* with actual value *other*
True negatives (TN)	Instances classified as *other* with actual value *other*
False negatives (FN)	Instances classified as *other* with actual value *y*

The fraction of relevant retrieved instances among all retrieved instances is called *precision*. The fraction of relevant retrieved instances among all relevant instances is called *recall*.

$$\text{Precision} = \tfrac{TP}{TP+FP} \qquad \text{Recall} = \tfrac{TP}{TP+FN}$$

The *F-measure* is the harmonic mean of precision and recall for each attribute value. The weighted average F-measure for all attribute values yields a more accurate evaluation of the entire model (Witten et al., 2016).

$$\text{F-measure} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.5}$$

To get an average F-measure for several class values, the F-measure for each value is weighted with the fraction of its occurrence over the total number of class instances.

$$\text{Weighted average F-measure} = \frac{\sum(F - measure_k * n_k)}{N} \tag{2.6}$$

Where n_k is the number of k values, $F-measure_k$ for attribute k and N is the total number of values. The average weighted *F - measure* is used as a primary evaluation metric in this thesis.

2.3 CRISP-DM

Cross Industry Standard Process for Data Mining (CRISP-DM) is a data mining process model developed by IBM. From an analysts perspective, CRISP-DM acts as a common process model providing guidance throughout the process, most importantly in terms of documentation and result evaluation. The model breaks the process into six phases: business understanding, data understanding, data preparation, modeling, evaluation and deployment (Wirth and Hipp, 2000). This thesis was conducted in sprints with each sprint incorporating the CRISP-DM model.

Chapter 3
Approach

3.1 Utilities

Together with our supervisors, we decided to utilize any data mining tool and/or code available and suitable for this thesis in order to find the best possible answers to our research questions rather then devote our time to rewrite existing code ourselves.

3.1.1 Waikato Environment for Knowledge Analysis

Weka is a collection of machine learning algorithms for data mining tasks. Written in Java, Weka offers the possibility to apply machine learning algorithms to datasets via a GUI or directly in code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization but can also be used to develop new machine learning schemes (Frank et al., 2016).

3.1.2 Osmosis

Osmosis is an command line Java tool for processing OSM data. It allows chaining components to perform larger operations (OpenStreetMap Wiki, 2017b). We use this tool mainly to pre-process large datasets and remove all unnecessary data in order to minimize the processing power required to run data mining/machine learning algorithms.

3.2 Geographical data extraction

Overpass Turbo (OpenStreetMap Wiki contributors, 2017) was used to extract small amounts of data at the initial stage of this thesis as it does not require additional software to be installed. Overpass Turbo runs any Overpass API (OpenStreetMap Wiki, 2017c) query and serves custom selected parts of the OSM map data up to ~300 MB. This theoretical limit has proven to fluctuate significantly between different browsers, machines and query parameters. For larger sets, we used binary .pbf data downloaded from Geofabric (Geofabrik GmbH, 2017) due to its superiority over OSM XML and JSON formats being smaller and faster to process. The data was later processed using Osmosis (OpenStreetMap Wiki, 2017b) providing tools to filter the data before generating an OSM XML file thus further reducing the amount of processing power necessary to compute the desired output.

3.3 Data analysis

Although all OSM elements follow the same scheme in terms of architecture, there are big differences in associated features and feature patterns. Elements can be categorized by introducing abstract classes through selection of most significant key/value pairs. An example of such category are elements associated with one of eight principal *highway* tag values for the road network, commonly called *roads*.

Such an organization makes it possible to expect elements containing the most significant feature with a certain value also to contain a set of key/value pairs necessary to give a complete description of the element. A majority of main road elements such as motorways will, apart from the *highway* tag, contain attributes such as *maxspeed*, *lanes*, or *ref*.

To minimize the diversity of both attribute keys and values, we chose to begin with the primary road network in Sweden. We chose Sweden in order to take advantage of our prior knowledge about the Swedish infrastructure, laws, and regulations. The dataset contains a relatively (to other categories) small number of different key values thus decreasing the number of missing values and minimizing the need of data imputation. Additionally, the elements associated with the *highway* tag key provide the basis for routing services making their accuracy of significant importance. The raw dataset consisting of all road elements in Sweden constitutes the starting point of our work and is referred to as D_0.

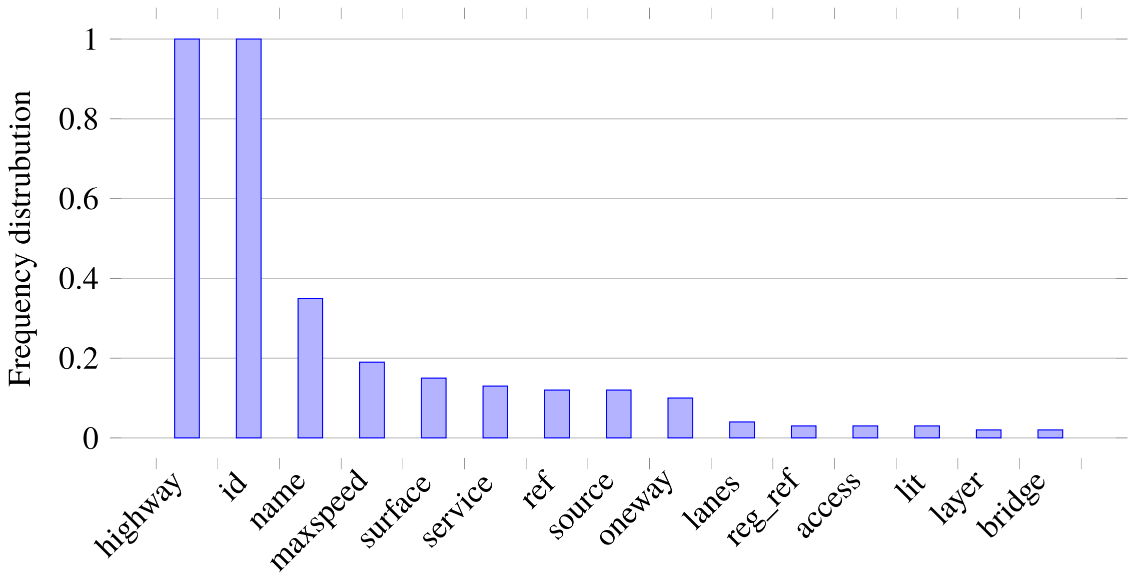
3.3.1 Data preparation

We began the data preparation process with developing generic parsers allowing us to select desired data from OSM XML files and generate an ARFF file necessary to process the data in Weka. We calculated the frequency distribution of each tag key among the 599,568 entities and removed all tag keys with a total appearance of less then 4%, see Figure 3.1. The 4% threshold was set arbitrary in order to reduce the computational power needed to compute a prediction model.

The preparation process was done manually by removing all super-attributes and attributes with long string values as statistical data analysis algorithms only accept nominal and numeric values. The process may seem trivial but there is more to it than first meets the eye. The most obvious way to identify a super-attribute or a long string is to look at the two following things:

- The number of different values for the attribute compared to the number of attribute occurrences (*DV/O*).
- The character length of the attribute value.

Figure 3.1: Tag key distribution. The *highway* and *id* are the only tag keys are present in all instances.



If the attribute has unique values for each instance, as in the case of ids, the *DV/O* ratio is 1. To remove all unwanted attributes, a maximal threshold can be set for both *DV/O* ratio and the total number of characters. The difficult part here is setting proper thresholds and even if one manages to find optimal values for the two, the method will still remove some attributes that may otherwise prove useful. The attribute *length* has a *DV/O* ratio of 0.304. For comparison, attributes *ref* and *maxspeed* have *DV/O* ratios of 0.142 and 0.0002 respectively. By setting a threshold of 0.1, the otherwise valuable attribute *length* would be removed.

In addition, there are three more kinds of unwanted attributes:

- Temporary attributes
- Weak attributes
- Malformed/wrong/misplaced attributes

The *notes* attribute is a prime example of a temporary attribute. It is used to supply extra information to other OSM users about roadwork, uncertainties and trivia. Weak attributes are attributes holding no relevant information about the instance itself in terms of classification. These can be *names*, *notes*, *created_by* tags etc. The value of a weak tag (or the tag itself) can be changed/removed at any moment with very little to no effect on other instance properties. Finally there are tags that do not apply within the guidelines of OSM. *highway_1*, *note_2*, *Lillvägen* are examples of such attributes being a result of mistakes and lack of knowledge. We manually removed all attributes matching the above criteria obtaining the dataset referred in this paper as D_1.

3.4 Data modeling

To be able to detect aberrations in the dataset, we chose to solve the problem by constructing prediction models and comparing the obtained results with values present in the dataset. By constructing a prediction model for a certain type of road, we use all the other attributes to classify the type of the road. If the predicted road type differs from the actual road type specified in the given instance, the aberration can be highlighted as an potential error. The substantial amount of missing data in the OSM dataset proved to be the biggest issue in classifier construction. During our initial experiments with the raw dataset, we concluded that all tested classification algorithms perform similarly, see Table 3.1. In order to grant ourselves the best preconditions for feature engineering, we chose decision trees as our primary classifier due to its natural model representation.

Table 3.1: Raw data performance - D_1. Train on 66% and test on 34% of the dataset.

Algorithm	Hyperparameters	Classifier	Accuracy	W.a. F-M
J48	C-0.25 M-2	Decision tree	38.5%	0.259
Naive Bayes	batch_size = 100	Naive Bayes	40.7%	0.298
JRip	folds = 3, min_w = 2.0	Rule induction	39.7%	0.277
IBk	n_neighbours = 10	k-nearest neighbour	43.5%	0.344

3.4.1 OSM incompleteness

Although most algorithms are able to handle missing values to some extent, the output quality improves as the number of missing values decreases. In Section 2.1.4, we discuss different generic data imputation techniques. The methodology works under the assumption that missing values cannot be computed in any other reliable way and that the missing value itself is a lack of information. This however, is not always the case with OSM data. The absence of the *maxspeed* tag rarely indicates unregulated speed limits. The information can be obtained by looking at the position of the studied element and its relation to other elements. A residential road in Sweden without any *maxspeed* tag can (in practise) inherit the value from several sources:

- Local regulations – Communities in Sweden are free to introduce their own traffic regulations. These are often displayed at every entrance point to the community/area (Transportstyrelsen, 2017).

- Municipal regulations – Municipalities are free to introduce their own traffic regulations, displayed at city entrance points.

- Traffic laws – In absence of any additional regulations, attribute values such as speed limit can be derived from traffic laws.

This property of the data makes the results of imputation using common techniques uncertain as it is hard to verify if the absent value should indeed be considered missing or can be derived by looking at additional information. Although the sources mentioned above are publicly available, the information requires manual imports where OSM lacks a connection between different levels of geographical entities. A randomly chosen node cannot be traced back to being a part of a larger entity unless it is a part of the boundary polygon defining the borders of the area. Connecting a way to a municipality therefore requires checking if the way is located within the borders of the municipality.

3.4.2 Implications

To address the ambiguity of missing data points, we first examine the correlation between different sets of attributes. The absence of a tag does not always imply incompleteness in an element description. On the contrary, OSM data is built on implications. The presence of one tag implies other tags pairs without the need of attachment.

$$\text{key=highway value=any} \implies \text{key=access value=yes}$$

As illustrated above, the presence of the highway tag, regardless of its value, implies the public accessibility of the road without the need of adding the tag $access = yes$. This is especially problematic as the implied tags are not present in the dataset and thus cannot be accounted for when performing classification. Implication rules are stated in the OSM Wiki and we did not manage to find any summarized dataset suitable for our purposes. As is often the case with implications, they seem obvious to humans but are hard to translate into Boolean rules unless there are clear patterns within the data itself. With no time to develop a Wiki parser, we made the assumption that all essential implications can be derived from patterns in the data.

3.4.3 Imputation using association rules

In our endeavour to decrease the number of missing values, we tried to use mean substitution and distribution based imputation mentioned in Section 2.1.4. Both techniques were very resource heavy and not tenable for future use. In the search for a more suitable solution, we tried association rule based imputation. Using the Apriori algorithm we derived association rules with a minimum support of 0.1. All rules with $confidence$ below 1 were discarded. The antecedent half of the remaining rules appeared in the consequent half in all derived instances thus confirming the rule to be true in all observations. An example of a obtained rule is shown in Table 3.2. Despite our attempts to combat missing

Table 3.2: Association rule example.

Rule	**conf**	**lift**	**lev**	**conv**
$\{maxspeed = 110\ , highway = motorway\} \Rightarrow \{oneway = yes\}$	1.0	1.01	0	2.02

values, all techniques applied to this point resulted in heavy biases and no performance improvements, for more see Section 4.1.

3.4.4 Presence labelling

To overcome the biases introduced upon imputation, we had to redefine the definition of incomplete data. Missing data points were given a default value of *missing* thus exploiting the possibility that the absence of a point could itself hold relevant information about the instance.

To evaluate how well the sheer existence of an attribute key describes an instance, the original dataset D_0 was transformed into a pure Boolean dataset D_2. All tag values i.e. the labels were replaced with values *present* or *missing*. The new dataset did not contain any missing values thus eliminating the need of imputation entirely. Eliminating the resource consuming process of imputation, we were able to utilize the entire dataset and all the attributes despite low value frequencies. The lack of attribute values made it possible to construct a model built on correlation between different tag keys by their co-existence with other keys.

3.4.5 Missing value labelling

To construct a model both capable of predicting tags keys and their values, we improved the solution used for D_2. All missing values in the new dataset D_3 were replaced with the value *missing* while all present values were kept unaffected. This allowed us to derive more complex patterns based on both actual attribute values and the synthetic value *missing* for absent data points.

3.5 Feature engineering

After exploiting the most suitable modelling method, we proceeded to feature engineering with the goal of further improving the classifier performance. Feature engineering is a fundamental process in machine learning as it makes algorithms more accurate, efficient and effective. It is also one of the most difficult tasks as it requires extensive knowledge not only about the utilized algorithms but also the domain itself. To compose a set of viable features we:

- Investigated the domain and analyzed possible feature candidates.
- Identified how each candidate can be represented as a feature.
- Introduced one feature at a time into the data.
- Evaluated the effects of a introduced feature.
- Kept or discarded the feature depending on the impact.

3.5.1 Synthetic attributes

As explained in Section 1.2.1, each element contains more information encapsulated in each node. Using each element's metadata such as geographical properties, we introduced a number of *synthetic* attributes. The final dataset D_4 is an extension of D_3 with all synthetic attributes described below.

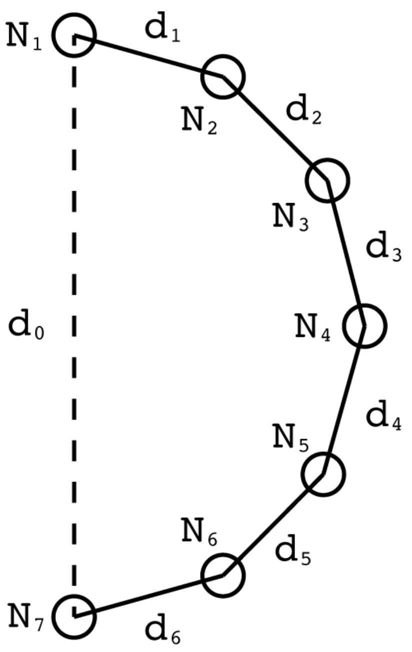


Figure 3.2: A simplified road element with nodes N_i and distances d_i between them.

Node count

The *node count* attribute is the sum of nodes N_i contained in each element, see Figure 3.2.

Element length

The *length* attribute is the sum of distances d_i between each consecutive node in an element expressed in meters, see Figure 3.2. It is calculated using the Haversine formula as the road elements in Sweden range between $0 \leq l \leq 90{,}000$ m with a mean of 507 m and are sufficiently distant from the earth's poles.

Curvature difference

The *curvature difference* is the fraction of an elements total length $d_1 - d_n$ over the distance between the first and the last node d_0, see Figure 3.2. The curvature distance is used to express a roads curvature in order to help the classifier to distinguish between e.g. a straight road and a roundabout. Once again, the Haversine formula is used to calculate all distances.

$$curvatureDiff = \frac{\sum_{i=1}^{n} d_i}{d_0} \tag{3.1}$$

Mean distance

The *mean distance* attribute is the mean distance $\mu(d)$ between all nodes in an element.

Mean density

The *mean density* attribute is used to determine if elements are located in a densely populated area. The density of an element is the mean value of all densities for each node in the element. The node density is measured by counting all nodes within a 1000 m x 1000 m square as illustrated in Figure 3.3.

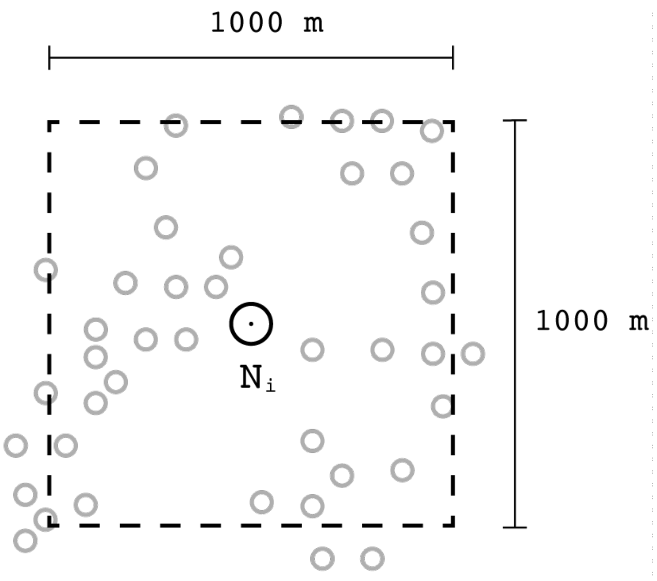


Figure 3.3: Node density calculation for node N_i.

3.5.2 Attributes based on neighbouring elements

To evaluate the consistency within consecutive elements, each element is compared to all elements connected to it i.e. sharing one or more nodes. Each tag associated with the evaluated element is compared to the same tag in neighbouring elements as seen in Figure 3.4. Absent tag keys and different values are treated equally – as an inconsistency.

3.5.3 Attribute selection

The main dataset used to train all classifiers contains a vast amount of features, many of which holding no relevant information. C4.5 prunes the tree by comparing the upper bound of the error rate confidence interval before and after removal of a node. This post-pruning approach computes the perfect tree and compares the pruning results for every leaf node parent in the tree from bottom to top. The process is very resource consuming as all data is used while constructing the initial tree and iterating it node by node. Furthermore, the large amount of features proved the pruning process alone insufficient as the obtained trees

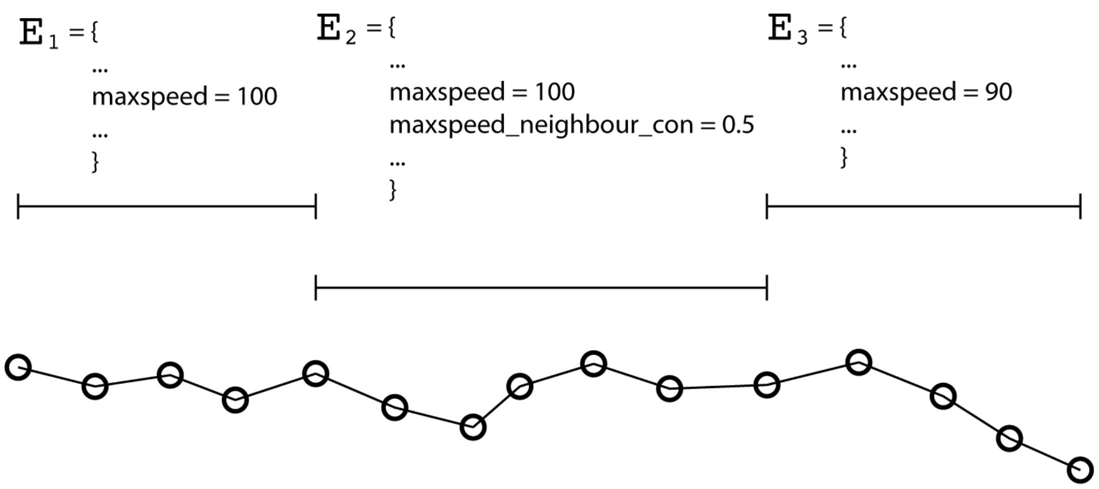


Figure 3.4: Simplified illustration of a neighbour data consistency evaluation. The *maxspeed_neighbour_con* tag in E_2 is given the value 0.5 as the *maxspeed* tag in E_1 and E_3 equals the value of E_2 in one of two connected elements.

still contained huge amounts of nodes due to the noisy nature of our data. To combat these issues, we introduced filters to the original dataset, one for each classifier. By computing the information gain for each attribute, we were able to significantly reduce the size of datasets (and thus the trees) used for training with no negative impact on the performance.

A filter for an attribute classifier is created by computing the information gain for all other attributes and saving only those with a satisfactory gain. The threshold was set to 0.001 meaning that all attributes even slightly decreasing the entropy after a split are kept while the rest is discarded. The filters are saved together with the classification models and applied to the main dataset upon usage.

3.6 Model evaluation

In order to measure the performance of all 600+ models we introduce custom weights to the arithmetic mean of *F-measure* for all models. Although the obtained value itself is not a direct representation of the *F-measure* for the entire system, it proved useful when comparing results upon changes in datasets, features, parameters, etc.

$$Weighted\ Average\ F - measure_{all} = \frac{\sum_{i=1}^{n}(tot\_ins_i - missing_i) * F - measure_i}{\sum_{i=1}^{n} tot\_insi - missing_i} \qquad (3.2)$$

Attributes often missing a value are assigned lower weights as the information provided by these attributes is less reliable as compared to attributes more commonly found in the dataset.

3.7 Model construction and classification

The classification models are constructed as shown in Algorithm 1. Building a classifier with sufficient accuracy around certain attributes may prove both impossible and meaningless. An example of such attributes are user annotations and temporal notes. The lack of patterns leading up to specific values of these attributes will result in poor performance. All models with a *F-measure* below *fM_thresh* are therefore discarded.

The classification of each attribute is done as shown in Algorithm 2. The provided input is a vector of all attributes in the element.

```
Data: Dataset $D$.
Result: A classification model for each attribute with weighted average F-measure
        $\geq 0.7$.
$iG\_thresh \leftarrow 0.001$
$fM\_thresh \leftarrow 0.7$
Remove attributes $id$, $ref$ & $name$ from $D$
forall attributes $att \in D$ do
    temporary dataset $tD \leftarrow D$
    forall attributes $a \in D$ do
        if $a \neq att$ then
            InformationGain $iG$ for attribute $a$
            if $iG < iG\_thresh$ then
                $tD \leftarrow tD - a$
            end
        end
    end
    Training set $trS \leftarrow 0.44 * tD$
    Test set $tS \leftarrow tD - trS$
    trS.classAttribute $\leftarrow a$
    tS.classAttribute $\leftarrow a$
    $m \leftarrow$ Classification model trained using J48 for attribute $att$ on $trS$
    $fM \leftarrow$ Weighted average F-measure for $m$ using $tS$ as test set
    if $fM \geq fM\_thresh$ then
        Save model $m$
    end
end
```

Algorithm 1: Attribute classifier construction

3.8 User interface

Oftentimes when constructing and reviewing ML models, it is hard to assimilate the results and demonstrate real life applications of the developed software. We felt it important to visualize and embody the results of our findings in a simple use case scenario. For this purpose, we developed a basic API enabling us to make prediction requests for single attribute values. The API takes an instance vector as input and returns an array of predicted

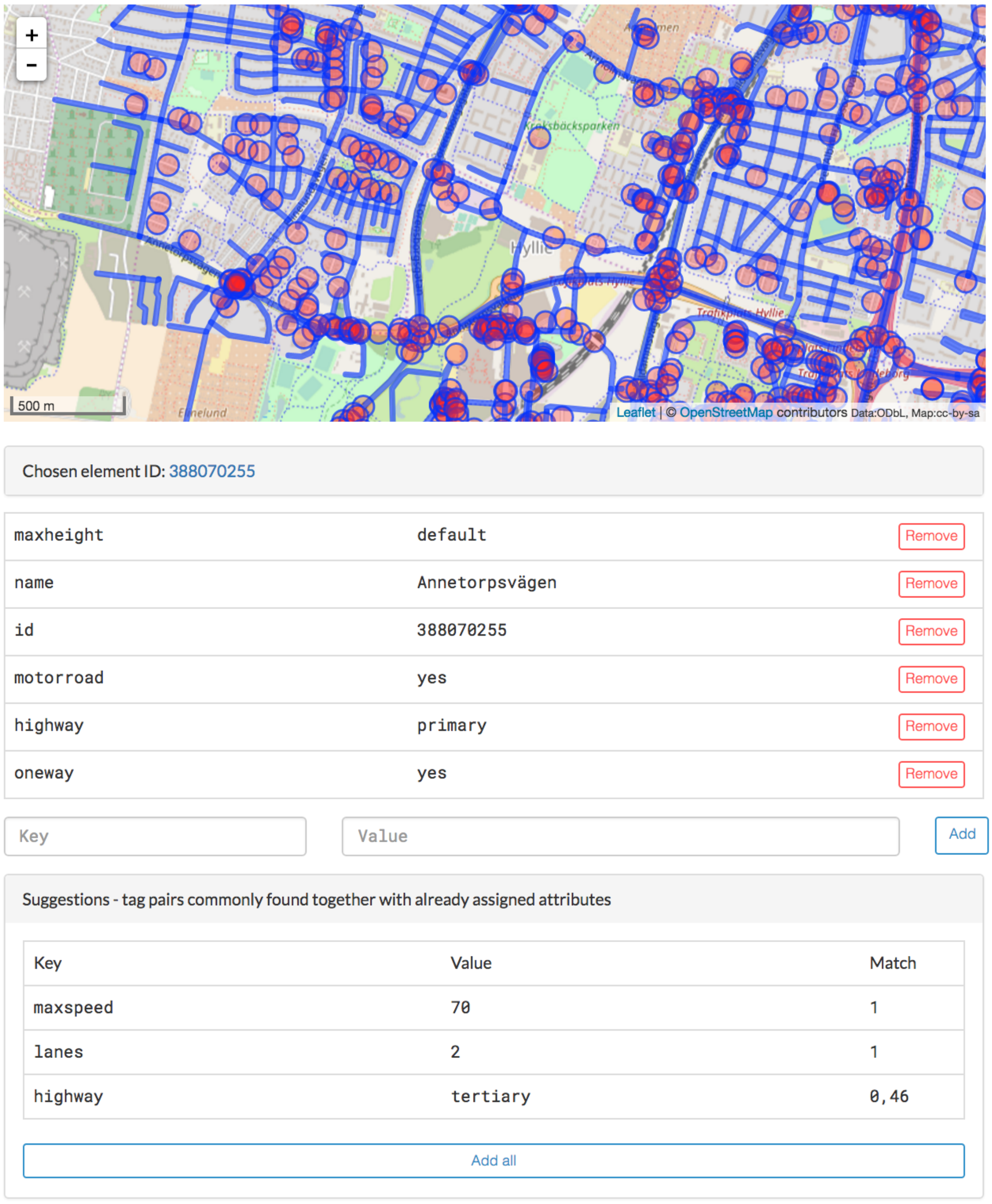
```
Data: Instance $I$.
Result: $suggestionList$ with $suggestions$ containing predicted value $v$ and
        probability $p$.
forall $attributes\ att \in I$ do
    load model $m$ for $att$
    double[] $probabilityForEachValue$ = $m$.distributionForInstance($I$);
    int $index \leftarrow$ index of max value in $probabilityForEachValue$
    if $index \neq att.index$ then
        $suggestion.v \leftarrow I$.value($index$);
        $suggestion.p \leftarrow probabilityForEachValue[index]$;
        $suggestionList$.add($suggestion$);
    end
end
```

Algorithm 2: Using classifier for suggestions

values for the class attribute together with a certainty percentage.

The demo application is a web application map editor, connected to OSM via an overpass turbo iframe. The editor stages a scenario in which the user is editing or inserting new map data. By selecting an element, the interface fetches all present attributes for the instance and requests value predictions for each attribute as described in Algorithm 2. If the predicted value differs from the actual attribute value, the web editor suggest a change. In cases where a single input vector returns several possible values, the most probable values are presented first, see Figure 3.5.

The editor is run on a local NodeJS server and the interface is a combination of React Redux and Bootstrap. The API requires all models to be pre-built and stored locally as it is deployed locally as a Java web server.



Chosen element ID: 388070255

maxheight	default	Remove
name	Annetorpsvägen	Remove
id	388070255	Remove
motorroad	yes	Remove
highway	primary	Remove
oneway	yes	Remove

Key Value Add

Suggestions - tag pairs commonly found together with already assigned attributes

Key	Value	Match
maxspeed	70	1
lanes	2	1
highway	tertiary	0,46

Figure 3.5: The editor demo user interface. The input data supplied to the API is displayed above the suggested values. The suggestions are whatever is returned by the model given the input vector. As the input changes upon edition, a new query is passed to the model in order to get updated predictions. The match value tells the user how reliable the prediction is according to the model, ranging from 0 to 1 where 1 implies the highest degree of certainty.

Chapter 4
Results

4.1 Imputation

Table 4.1 shows the confusion matrix for the D_0 dataset after removing all instances with missing values. This approach is of course not applicable for the main dataset as such filtering would remove all instances in the dataset. No instance in the unfiltered data contains a non-missing value for every possible attribute. The results in Table 4.1 are shown for the sole purpose of illustrating the effects of imputation.

Table 4.1: Confusion matrix for the highway tag J48 classification using dataset D_0. All instances with missing values removed. Bold highlight on most frequent prediction.

a	b	c	d	e	f	g	h	<–	classified as
0	27	8	0	**79**	0	0	14	|	a = unclassified
1	**160**	17	0	65	0	0	3	|	b = residential
1	55	37	0	**535**	0	0	82	|	c = tertiary
0	3	0	**898**	78	0	0	317	|	d = motorway
0	44	13	9	**1095**	0	0	241	|	e = secondary
0	13	0	0	6	**29**	0	0	|	f = service
0	2	4	20	257	0	0	**509**	|	g = primary
0	0	13	300	413	0	0	**1764**	|	h = trunk

The results of the default distribution based imputation used in J48 are seen in Table 4.3. As seen in Table 4.2 instances with class labels unclassified, residential and service account for over 83% of all data. These elements also have the lowest average number of associated tags meaning most attributes in these instances have no (and should not have any) value. By predicting almost all attributes to be missing, the classifier introduces a bias towards labels having no or very few attributes other than the class attribute.

Table 4.2: *highway* class distribution in all datasets together with average number of non-missing tags associated with each label.

Class label	Count	Fraction	Average number of tags
unclassified	114537	0.191	0.779
residential	180933	0.302	0.770
tertiary	51985	0.087	1.673
motorway	4211	0.007	5.200
secondary	25053	0.042	2.434
service	203476	0.339	0.779
primary	8405	0.014	2.492
trunk	10968	0.018	4.374

Table 4.3: Confusion matrix for the *highway* tag J48 classification using dataset D_0. Bold highlight on most frequent prediction.

a	b	c	d	e	f	g	h	<–	classified as
0	895	0	0	0	**38002**	0	0	|	a = unclassified
0	9712	0	0	0	**51832**	0	0	|	b = residential
0	608	0	0	0	**17045**	0	0	|	c = tertiary
0	1	0	0	0	**1491**	0	0	|	d = motorway
0	134	0	0	0	**8449**	0	0	|	e = secondary
0	376	0	0	0	**68768**	0	0	|	f = service
0	14	0	0	0	**2794**	0	0	|	g = primary
0	13	0	0	0	**3719**	0	0	|	h = trunk

Association rule imputation reduced the amount of missing data significantly as seen in Table 4.4. Despite this, the imputation method had a negative impact on the classifier performance, see Table 4.11. The strong bias introduced in distribution based imputation was weakened by association rule imputation as seen in Table 4.5. The classifier makes predictions for more than just the most frequent labels but the bias is still tangible where most predictions result in labels unclassified, residential or service.

Table 4.4: Missing value fractions before and after imputation using association rules.

	D_0 (raw)	D_1 (Apriori imputed)
lanes	0.96	0.28
maxspeed	0.81	0.48
oneway	0.90	0.28

Table 4.5: Confusion matrix for the *highway* tag J48 classification using dataset D_1. Bold highlight on most frequent prediction.

a	b	c	d	e	f	g	h	<–	classified as
792	2305	42	0	9	**35747**	0	2	|	a = unclassified
146	6424	20	0	4	**54950**	0	0	|	b = residential
1422	4581	166	0	52	**11419**	0	13	|	c = tertiary
50	409	0	**972**	14	37	0	10	|	d = motorway
867	**4288**	168	6	224	2966	0	64	|	e = secondary
167	1080	4	0	0	**67893**	0	0	|	f = service
356	**1617**	14	25	39	513	0	244	|	g = primary
376	**1805**	29	289	51	290	0	892	|	h = trunk

Both standard imputation techniques and Apriori introduced a bias favoring instances with present data and applying the incorrect derivations to instances regarded as incomplete. Given the OSM dataset it was proven that missing values had to be accounted for without the assumption that absent data points had to be given a value present in the rest of the data.

4.2 Presence labelling

Substituting all attribute values with Boolean present and missing in D_2 outperformed both D_0 and D_1 as seen in Table 4.11. The main drawback of this approach is the inability to predict actual tag values. The important finding at this step was that sheer co-existence of attribute keys had a major impact on classification results. Furthermore, the approach allows classification training to be performed with minimal amounts of computational power as there is no need of imputation. Compared to D_0 and D_1, the classifier built around D_2 was able to form a pattern of correct classifications as seen in Table 4.6.

The inability to train the classifier on different tag values poses an issue clearly visible in classification results performed on similar instances with different class labels. Elements with class labels trunk and motorway are almost identical to each other in terms assigned key values. The main difference between the two lies in tag values where a motorway typically has higher speed limits and more lanes. Distinguishing between these elements is therefore impossible based on pure tag key presence and requires more information to be supplied in the training phase.

Table 4.6: Confusion matrix for the *highway* tag J48 classification using dataset D_2. Bold highlight on most frequent prediction.

a	b	c	d	e	f	g	h	<–	classified as
9481	7983	217	0	100	**21115**	1	0	|	a = unclassified
1870	**45333**	277	0	78	13985	0	1	|	b = residential
223	3633	**11690**	2	1603	270	193	39	|	c = tertiary
3	6	7	**720**	101	1	81	573	|	d = motorway
84	1041	3352	26	**3550**	55	425	50	|	e = secondary
2968	2329	19	0	7	**63819**	1	1	|	f = service
19	10	576	16	1046	5	**1077**	59	|	g = primary
3	6	141	313	327	5	437	**2500**	|	h = trunk

4.3 Missing value labelling

Substituting missing data with a *missing* label improved the classifier performance as seen in Table 4.11. More importantly, it also allowed to make value predictions together with predictions for existence. The latter is especially important as it makes the classifier output consistent with the OSM scheme. Considering all possible attributes in the dataset, only a small fraction is set in each element. The possibility to both predict a value of an attribute or its existence (whether the element should at all be associated with the attribute) gives predictions that follow the pattern found in the actual dataset. Issues with similar element structures in motorways and trunk roads mentioned in section 4.2 are resolved at this stage, see Table 4.7.

Table 4.7: Confusion matrix for the *highway* tag J48 classification using dataset D_3. Bold highlight on most frequent prediction.

a	b	c	d	e	f	g	h	<–	classified as
31522	23869	2206	1	537	10594	81	23	|	a = unclassified
9147	**78067**	1868	0	492	18762	25	0	|	b = residential
5654	6599	**14322**	4	2694	1563	275	74	|	c = tertiary
1	0	6	**2284**	60	6	52	194	|	d = motorway
1054	1877	3535	25	**7208**	453	782	196	|	e = secondary
6407	27205	650	0	157	**87632**	18	5	|	f = service
322	373	611	63	1214	83	**1974**	402	|	g = primary
36	57	139	351	430	9	572	**4909**	|	h = trunk

4.4 Synthetic attributes

D_4 is an extension of D_3 with custom features derived from metadata as described in Section 3.5.1. The introduction of custom features increased the weighted average F-Measure for the highway classifier by 0.145 as seen in Table 4.11. The biggest improvement however, is seen in Table 4.9. Despite the overall performance boost, the classifier at this stage is also capable of classifying less frequent values. The importance of all synthetic attributes is presented in Table 4.8. On average, every custom feature appears in the top 10 to 20 most significant attributes based on reduced entropy difference – the information gain. This of course is an expected result as the features were introduced in order to minimize the entropy with each split. Nevertheless, the results prove the suitability of our approach.

Table 4.8: Attribute information gain for class attributes highway and lanes.

Attribute	Info. gain
neighbour_highway	0.74338
maxspeed	0.34341
service	0.22883
totLength	0.22373
neighbour_maxspeed	0.19461
meanDensity	0.16035
neighbour_oneway	0.13328
lanes	0.12422
oneway	0.12369
nodeCount	0.11486
...	
curvatureDiff	0.09197
meanDistance	0.08896
...	

Attribute	Info. gain
neighbour_lanes	0.18072
highway	0.12422
maxspeed	0.11134
neighbour_oneway	0.09086
oneway	0.08141
neighbour_maxspeed	0.0669
curvatureDiff	0.01693
surface	0.01686
neighbour_surface	0.01037
meanDistance	0.00975
...	
meanDensity	0.0037
totLength	0.00325
...	

Table 4.9: F-Measure for different layer tag values using J48.

Tag values	D_2	D_3	D_4	Freq. distribution
Missing	0.998	0.998	0.927	0.9764
Present	0.899	-	-	0.0246
1	-	0.909	0.919	0.019
-1	-	0.852	0.857	0.0031
2	-	0.000	0.319	0.0006
-2	-	0.000	0.144	< 0.0001
0	-	0.000	0.494	0.0004
5	-	0.000	0.590	0.0001
...				
Weighted Avg.	**0.995**	**0.994**	**0.995**	

4.5 Attribute selection

Computing models using all attributes resulted in long computation times and a greater exposure to possible overfitting. As a countermeasure to these issues, we introduced attribute selection. Although overfitting can be reduced by increased confidence levels and

Table 4.10: Confusion matrix for the *highway* tag J48 classification using dataset D_4. Bold highlight on most frequent prediction.

a	b	c	d	e	f	g	h	<–	classified as
42985	15803	1456	1	281	8288	16	3	|	a = unclassified
10958	**86579**	1606	0	229	8991	2	6	|	b = residential
3030	3342	**21659**	7	2471	254	409	13	|	c = tertiary
3	2	5	**2468**	44	0	81	0	|	d = motorway
389	633	5682	34	**7195**	3	1191	3	|	e = secondary
1943	845	23	0	0	**119243**	0	20	|	f = service
25	19	826	70	1173	0	**2929**	0	|	g = primary
10	4	1	0	0	40	0	**6448**	|	h = trunk

Table 4.11: F-Measure for different highway class values using J48 using all datasets derived in this project.

	D_0	D_1	D_2	D_3	D_4
Unclassified	0.000	0.037	0.338	0.362	0.671
Residential	0.265	0.153	0.575	0.745	0.803
Tertiary	0.000	0.018	0.474	0.692	0.694
Motorway	0.000	0.698	0.534	0.575	0.952
Secondary	0.000	0.050	0.373	0.474	0.543
Service	0.526	0.559	0.594	0.760	0.921
Primary	0.000	0.000	0.175	0.444	0.606
Trunk	0.000	0.360	0.713	0.721	0.992
Weighted Avg.	**0.259**	**0.258**	**0.516**	**0.655**	**0.800**

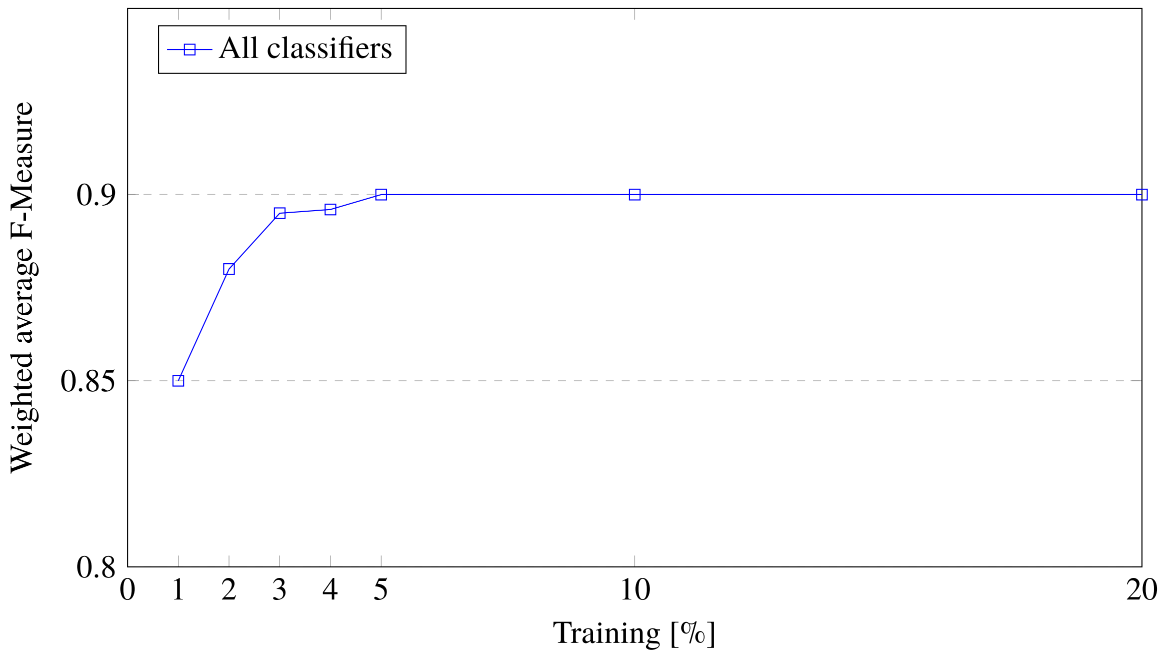
minimal number of objects in each node, it does not have any positive impact on the computation time. By introducing filters with minimal information thresholds, we were able to maintain the same performance while reducing the amount of attributes in the training data significantly. As an example, the *highway* tag classification tree had over 240,000 nodes before attribute selection and roughly 29,000 after.

4.6 Learning curve

The overall performance of all classifiers is difficult to summarize in one value. As mentioned in Section 3.6, we developed our own metric in order to get a reference point during optimization. The learning curve shown in Figure 4.1 shows performance stabilization with training on about 5% of the data. This means the data contains many entities from which the same classification rules can be derived thus decreasing the data amount necessary to compute the classification model. The graph appearance is motivated as follows:

1. y_min = 0.8 – The noise cap of the used data is ~0.8 due to the large fraction of missing values. By predicting the value missing on all attributes, the weighted average F-Measure will settle on the noise cap. This implies that the data overall is poor

Figure 4.1: Weighted average F-Measure for all classifiers



in populated features witch is consistent with our initial findings.

2. x_max = 20 – As the percentage of training data increases, the results are unaffected thus eliminating the need of further illustration. The main purpose of the graph is to illustrate the “knee” where the performance stabilizes.

Chapter 5
Discussion

The initial objective of this thesis was to derive patterns from attributes (and their values) by studying possible correlation given by occurrence and value dependency. We thought of the solution as a semantic correlation network rather than a feature based classifier. As we investigated the domain and previous efforts to achieve goals similar to ours, we realized that this approach will not be able to provide a feasible solution. The attribute data itself holds very little information and does not include the structural and positional properties of an element. It was obvious to us that feature engineering would be necessary and that it would have a crucial impact on our success. We began the process by identifying biggest obstacles preventing us from using machine learning classifiers on the OSM dataset. From there, we worked in sprints following the CRISP-DM model, gradually working our way to a satisfactory result.

5.1 Data selection

During the initial stages of this thesis project, we wanted to develop a generic system compatible with any OSM entity. Discovering the nature of OSM data and the diversity of features forced us to narrow our scope. In collaboration with our supervisor, we chose to work with roads due to their importance and suitable extent. Due to the limited time scope of this thesis, we decided not to set up development environments on powerful computers or remote servers. We wanted to be able to develop and continuously evaluate the performance of our software on our own laptop computers. The approach forced us to keep things light, efficient and effective as we had modest amounts of computational power. To meet these requirements and keep the computation time at a reasonable level, we restricted our target map area to the territory of Sweden. The motivation behind choosing Sweden was our prior knowledge of Swedish laws and regulations. This did indeed save us some time but was the partial reason to many preparations as Sweden does not have the best OSM data compared to other countries. Densely populated countries tend to have better data meaning better prerequisites for statistical analysis. Going for a better mapped region of Europe e.g. Germany would give us more time to focus on classifier construction rather than data pre-processing.

5.2 Data preparation

The subject of initial feature selection is an important part of this thesis. Due to the reasonably low number of different features and the issues mentioned above, we removed all unwanted features manually. In order to make the entire process generic, especially for large datasets, this procedure would require an automated solution.

5.3 Machine learning process

5.3.1 Data modelling

The large amount of missing values proved to be the biggest obstacle during initial classification tests. A majority of entities in the dataset share only one tag – *highway*. This made classification difficult and stated the need of additional features.

5.3.2 Feature engineering

Prior to the introduction of synthetic attributes, more than 80% of the data had an average of less than one associated tag per instance, see Table 4.2. By adding custom features to all instances, the average increased with the number of added attributes. The machine learning algorithm now had the possibility to learn the difference between elements even if they were poorly tagged in the raw dataset.

Our primary goal when developing new features was to include both the physical properties of an element, its position, and relation to other elements. To transform information about an element's position into specific features, we started by dividing Sweden into tiles based on geographical coordinates. The method has a big advantage as it can utilize region specific information such as divergent speed limits in different cities. Eventually, we abandoned this approach as we ran into issues with elements overlapping several tiles and the inability of deriving region specific information from OSM data. The mean density attribute is a simple substitute used to describe the type of environment the element is positioned within. It is not tied to any particular region but discloses the type of surrounding – dense regions for urban environments, sparse for countryside. This can be improved by expanding the reference area, checking neighbours neighbours and other elements in close proximity.

The main issue with synthetic attributes is that they need to be calculated for each element. The computation cost grows exponentially as the amount of data increases. Calculating the mean density for each node, being the most expensive attribute to obtain, has a time complexity of $O(n^2)$. Furthermore, any changes made in one element require recalculation of values in every affected element.

5.3.3 Optimization

In our endeavour to create as resource effective system as possible, minor optimizations were made along the entire process. Code efficiency optimization, database indexing, volatile memory storage and multithreading are examples of measures taken to ensure fast data pre-processing, data selection and model computation. The biggest breakthrough however was attribute selection. It decreased the average computation time by over 8000% by drastically decreasing the number of attributes used in the training process.

5.3.4 Model evaluation

When we started to create multiple classifiers, one for each tag key we started to evaluate the performance by calculating the mean value of all classifiers *Weighted Average F-Measure*. The problem with this approach was that the score never averaged below 0.99 due to the fact that most of the classifiers simply guess *missing* on all instances. To improve the evaluation of our models we needed a value accounting for the amount of missing values. We therefore introduced *Weighted Average F-measure*$_{all}$ explained in Equation 3.2.

5.3.5 Hyperparameters

Having one classifier for each attribute made it hard to tweak and evaluate the input parameters for the J48 algorithm. We did try different settings but the performance improvements were insignificant. Although we did not pursue this topic, we can recommend it for future improvements aiming to get the last bit of performance out of the algorithm.

5.4 User interface

Aside from being helpful while explaining how the models work and to visualize a possible use case, the user interface helped us to improve the system. During the development of the UI and its API, we repeatedly found out the models were giving unexpected predictions. Sometimes there was an error in model creation or use, sometimes the models gave good numbers but the prediction *looked* wrong at the first glimpse, encouraging further investigation. This helped us further develop the models, introducing more synthetic attributes and the weighted model evaluation.

5.5 Alternate use cases and future work

While the use case presented by us demonstrated an editing scenario, the system can also be used for plain error detection. By comparing the actual values in each instance with values predicted by the models, potential aberrations/errors can be discovered.

Given the time, resources, expertise and skills available to us while conducting this thesis, we came up with a system we argue is a suitable solution to the research questions. Nevertheless, there are several things that can be improved, changed and further studied. Many enhancements have been previously mentioned in this report but we summarize these together with additional improvements below.

- Automatically remove *unwanted* attributes.
- Evaluate other classification algorithms.
- Examine the impact of hyperparameter optimization.
- Introduce more synthetic attributes.
- Import data from other sources.
- Train and evaluate models on larger datasets, for example all roads in Europe.
- Expand our method to include other types of OSM entities.
- Explore new modelling approaches upon dataset expansion – both in terms of features and size.

Chapter 6
Conclusions

OSM is without doubt, one of the most prominent collaborative projects. Developed by the community for the community, it does not only give free access to a worldwide map but also enables any interested party to contribute with their own data. While crucial for the project itself, this open approach makes it difficult to enforce and maintain good data quality.

In this thesis, we propose what we argue is a possible solution for semantic accuracy improvement in OSM data using machine learning techniques. Using statistical analysis, we construct prediction models for each attribute and compute a suggested value to either prove or disprove the validity of the current value or suggest the value of an absent attribute. It was proven that the raw data itself given its poor quality in terms of coverage and accuracy is not sufficient to construct a reliable model. The quality was improved by filtering out noisy or irrelevant attributes and adding custom features based on entity properties. The derived prediction models are based on the J48 decision tree algorithm and achieve a *Weighted Average F-measure*$_{all}$ over 90%. Furthermore, it was proven that the quality of the derived propositions is highly dependent on data quality rather than the used algorithm or different hyperparameters.

To illustrate a possible use case scenario in which the system can contribute to a higher semantic accuracy in OSM data, we created an end-to-end application. The application simulates an editing scenario in which the user is given attribute suggestions while editing current/adding new data to the OSM dataset.

In summary, these are the most important findings discovered conducting this thesis:

- Data imputation using complete instances introduces heavy biases and weakens the classifier.

- Accounting for missing data, instead of removing or imputing it, improves classifier performance significantly.

- Despite its simplicity compared to other classification algorithms, the decision tree algorithm proved to be suitable for this task.
- Data pre-processing, modelling and feature engineering have a remarkably bigger impact on the results than different algorithms and hyperparameters. This particular finding is not unique to our project as numerous researchers have reached the same conclusion, see (Crone et al., 2006).
- Introduction of synthetic attributes is necessary to obtain satisfactory classification results.
- High attribute occurrence yields better predictions.
- It is possible to suggest changes during editing using models trained on current data.
- Only a small fraction of all data holds valuable information upon classification of each attribute.
- It is possible to find elements having bad or possibly wrong tags by checking if the predicted tag value and the actual value differ.

Bibliography

Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499.

Ali, A. L., Schmid, F., Al-Salman, R., and Kauppinen, T. (2014). Ambiguity and plausibility: managing classification quality in volunteered geographic information. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 143–152. ACM.

Basiri, A., Jackson, M., Amirian, P., Pourabdollah, A., Sester, M., Winstanley, A., Moore, T., and Zhang, L. (2016). Quality assessment of openstreetmap data using trajectory mining. *Geo-spatial information science*, 19(1):56–68.

Crone, S. F., Lessmann, S., and Stahlbock, R. (2006). The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing. *European Journal of Operational Research*, 173(3):781 – 800.

Frank, E., Hall, M. A., and Witten, I. H. (2016). *The WEKA Workbench. Online Appendix for "Data Mining": Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.

Geofabrik GmbH (2017). Geofabric – download openstreetmap data. `https://download.geofabrik.de/europe.html`.

Haklay, M. (2010). How good is volunteered geographical information? a comparative study of openstreetmap and ordnance survey datasets. *Environment and planning B: Planning and design*, 37(4):682–703.

Musil, C. M., Warner, C. B., Yobas, P. K., and Jones, S. L. (2002). A comparison of imputation techniques for handling missing data. *Western Journal of Nursing Research*, 24(7):815–829.

OpenStreetMap Wiki (2017a). Elements – openstreetmap wiki. `http://wiki.openstreetmap.org/w/index.php?title=Elements&oldid=1429253`.

OpenStreetMap Wiki (2017b). Openstreetmap wiki – osmosis. `https://wiki.openstreetmap.org/wiki/Osmosis`.

OpenStreetMap Wiki (2017c). Openstreetmap wiki – overpass api. `http://wiki.openstreetmap.org/wiki/Overpass_API`.

OpenStreetMap Wiki (2017d). Stats – openstreetmap wiki. `http://wiki.openstreetmap.org/w/index.php?title=Stats&oldid=1446588`.

OpenStreetMap Wiki contributors (2017). Openstreetmap wiki – overpass turbo. `http://wiki.openstreetmap.org/wiki/Overpass_turbo`.

Russell, S. J. and Norvig, P. (2010). Artificial intelligence (a modern approach).

Transportstyrelsen (2017). Svensk trafikföreskriftssamling – transportstyrelsen. `https://rdt.transportstyrelsen.se/rdt/defaultstfs.aspx`.

Vandecasteele, A. and Devillers, R. (2013). Improving volunteered geographic data quality using semantic similarity measurements. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1(1):143–148.

Wikipedia (2017). Openstreetmap – wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=OpenStreetMap&oldid=770582840`.

Wirth, R. and Hipp, J. (2000). Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, pages 29–39. Citeseer.

Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTATIONSDAG 2017-08-25

EXAMENSARBETE Improving the semantic accuracy and consistency of OpenStreetMap using modern machine learning techniques
STUDENTER Hannes Sandberg, Michal Stypa
HANDLEDARE Pierre Nugues (LTH)
EXAMINATOR Elin Anna Topp (LTH)

AI-assistent för kartläggning i OpenStreetMap

POPULÄRVETENSKAPLIG SAMMANFATTNING **Hannes Sandberg, Michal Stypa**

Över fyra miljoner bidragande användare och ett nära laglöst bidragssystem. Hur säkerhetsställs det att riktlinjer tolkas och implementeras rätt? I detta arbete används maskininlärning för att upplysa kartläggare om avvikelser, ofullkomligheter och direkta fel samt föreslå fler egenskaper.

Genom att förse maskininlärningsalgoritmer med data från över 600 000 vägar i Sverige har vi lyckats ta fram ett hjälpverktyg för kartläggning och felkontroll. Ta inmatning av ett nybyggt stycke av E4:an som exempel. Genom att kolla på en rad olika parametrar som liknande vägar, området i nära anslutning och vägens uppbyggnad föreslår systemet vägens mest sannolika egenskaper. Dessa skulle kunna vara en hastighetsbegränsning på 110 km/h, enkelriktning och två filer. Vid inmatning av orimliga egenskaper föreslår systemet lämpliga åtgärder. Skulle kartläggaren beskriva E4:an som ett objekt där fotgängare tillåts kommer systemet säga att det med största sannolikhet är fel.

Geografiska objekt märks i OpenStreetMap (OSM) med så kallade taggar. Dessa innehåller all information om objektet så som position, förhållande till andra objekt och en rad beskrivande egenskaper. Hur olika objekt ska beskrivas finns dokumenterat på OSM's hemsida men OSM låter användarna själva avgöra hur ett objekt ska taggas. Denna frihet gör att systemet är väldigt lättanvänt, särskilt för nya kartentusiaster, men medför skillnader beroende på kartläggarens tolkning av vad som är korrekt.

Detta brus gör det nästintill omöjligt att använda datan i analytiska sammanhang då datorer behöver tydliga regler. För att underlätta kartläggningsarbetet och framför allt uppmana konsekvent beskrivningsmetodik har vi tagit fram ett verktyg som bygger på befintliga mönster.

För att få fram dessa har vi använt maskininlärningsalgoritmer där datorn, utan vår hjälp, härleder mallar för hur olika objekt bör se ut. Detta åstadkoms genom att analysera samband mellan egenskaper och dess olika värden. Eftersom systemet bygger på statistisk analys kan även tidigare osedda objekt kategoriseras. Sambandsmallen mellan egenskaper ger även en procent på hur sannolikt ett befintligt värde är tillsammans med andra egenskaper samt hur sannolika eventuella förslag är.

Skulle samma problem vilja lösas utan artificiell intelligens skulle ett enormt regelverk behöva tas fram för hand där alla möjliga utfall täcks. Även då skulle systemet inte vara kapabelt att jobba med nyintroducerade objekt eftersom regelverket då bara skulle passa den ursprungliga datan.

Under arbetets gång har det framkommit att det enbart krävs 5% av alla vägar i Sverige för att systemet skulle kunna hantera vilken svensk väg som helst.