# Automatic Extrapolation of Missing Road Network Data in OpenStreetMap

**Stefan Funke** FUNKE@FMI.UNI-STUTTGART.DE
University of Stuttgart, 70569 Stuttgart, Germany

**Robin Schirrmeister** SCHIRRMR@INFORMATIK.UNI-FREIBURG.DE
University of Freiburg, 79110 Freiburg, Germany

**Sabine Storandt** STORANDT@INFORMATIK.UNI-FREIBURG.DE
University of Freiburg, 79110 Freiburg, Germany

## Abstract

Road network data from OpenStreetMap (OSM) is the basis of various real-world applications such as fleet management or traffic flow estimation, and has become a standard dataset for research on route planning and related subjects. The quality of such applications and conclusiveness of research crucially relies on correctness and completeness of the underlying road network data. We introduce methods for automatic detection of gaps in the road network and extrapolation of missing street names by learning topological and semantic characteristics of road networks. Our experiments show that with the help of the learned data, the quality of the OSM road network data can indeed be improved.

## 1. Introduction

OpenStreetMap (OSM) is a huge collection of crowdsourced spatial information. The goal of the OSM project is to map the whole world with all its road networks, buildings, regions and other kinds of natural and man-made entities. The OSM data set size increases significantly every year as more and more parts of the world are covered, and information becomes more detailed. For example, the world-wide road network in OSM contained at the beginning of 2007 less than 30 million data points whereas in 2013 this number has grown to more than two billions. Nowadays, the quality of OSM data often even exceeds the quality of proprietary data.

OSM is the basis for numerous applications and research projects, concerned e.g. with pedestrian and vehicle navigation (Holone et al., 2007; Vetter, 2010), location-based-services (Mooney & Corcoran, 2012), disaster warning (Rahman et al., 2012), fleet management (Efentakis et al., 2014), traffic estimation (Tao et al., 2012) and many other related topics[1]. Completeness of the road network data is mandatory for these applications to guarantee usability in practice. Road networks extracted from OSM (mainly Japan, Germany, North and South America and Australia) have also become standard benchmarks in route planning papers, see (Delling & Werneck, 2013; Baum et al., 2013; Funke et al., 2014). For research on route planning the completeness of the data plays an important role, as the developed algorithms are designed to take typical connectivity characteristics of road networks into account.

But while the OSM data is of very high quality already, there is still structural information missing, as e.g. road or path sections (see Figure 1). Moreover street names are far from being complete. The correct street name is necessary for specifying start and destination in a route planning query, for location-based services ('all shops on Norris Street') and for answering complex route planning queries like 'from A to B avoiding Park Street' accurately.

In this paper we design a classifier based on learning topological and semantic characteristics of road networks which can then be used to identify pairs of candidate locations where road segments are likely to be missing inbetween. We refer to missing structural data as *holes* in the following. Furthermore we show how to instrument machine learning techniques to identify road segments where the name tag can be extrapolated with high confidence. Our experimental results prove the ability of our methods to enhance the quality of OSM road network data considerably.

[1] http://wiki.openstreetmap.org/wiki/List_of_OSM-based_services

*Proceedings of the 2nd International Workshop on Mining Urban Data, Lille, France, 2015.* 

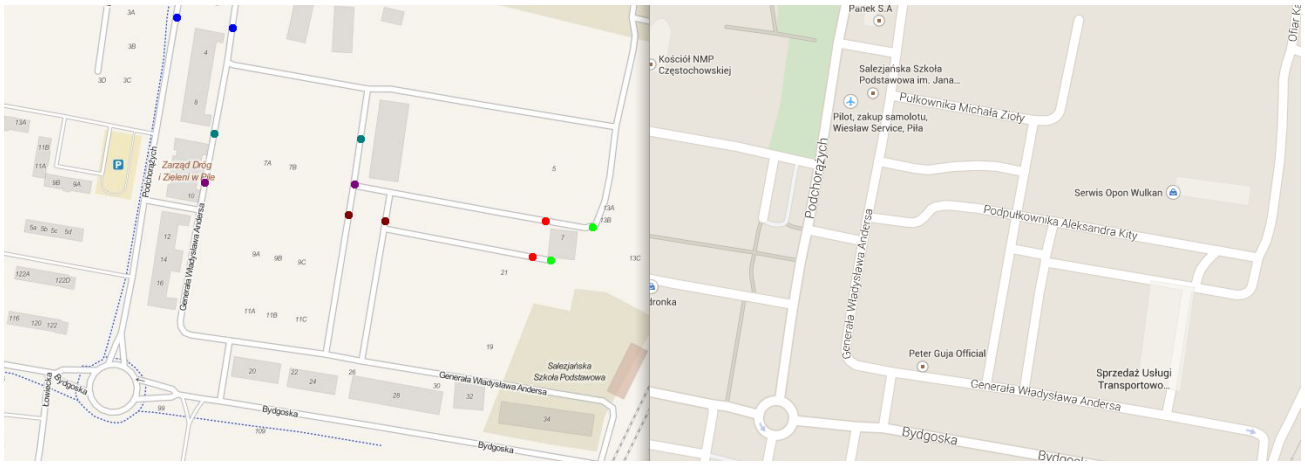


*Figure 1.* OSM based map (left) and GoogleMaps on a small cut-out of Poland. In the OSM map, the roundabout in the lower left corner is much more detailed and more building footprints and house numbers are available. For the coloured points in the left image, though, there are road segments missing as observable on the right.

## 2. Related Work

Studies on completeness and correctness of the OSM data are numerous, see e.g. (Girres & Touya, 2010) or (Haklay, 2010). The problem is that there either needs to be a ground truth one can compare to in an automated way (as investigated in (Fan et al., 2014) for building footprints in Munich), or data has to be manually compared to proprietary data. In both cases, the ground truth sample sizes are typically limited. Machine learning was applied to OSM data in order to automatically assess the quality of the road network data (Jilani et al., 2013b). Here, characteristics of certain street types (as e.g. motorways) are learned, including features such as total street length, number of dead-ends, number of intersection points and connectivity to other street types. In the quality analysis, feature vectors of streets are compared to the learned feature vector for the respective street type. If they do not resemble each other it is assumed that the data quality of the considered street is poor. The authors state that these learned features are also useful to classify streets with unknown type.

Preliminary results on automated quality improvement of OSM data were also reported in (Jilani et al., 2013a). Here, Artificial Neural Networks (ANN) are applied to distinguish residential and pedestrian streets; features include node count within a bounding box and betweenness centrality. In (Jilani et al., 2014), the automated street type classification for OSM was considered in more detail. In addition to the above mentioned features, the shape of the street is considered. About 20 different OSM street types were used in the experimental evaluation. The classification accuracy varies widely but for some types even an accuracy of 100% is achieved.

Further research on enhancing or correcting OSM data automatically (not necessarily using machine learning techniques) include the deduction of turn restrictions from GPS tracks (Efentakis et al., 2014), the detection of vandalism using a rule-based approach (Neis et al., 2012), or the identification of basic spatial units (so called parcels) for fine-scale urban modeling (Long & Liu, 2013).

To the best of our knowledge, tools for detecting missing parts of the OSM road network automatically were not investigated before, the quality assurance tools in the OSM project[2] mostly focus on detecting syntactic errors in the map specification. Hole detection in other kind of networks, as e.g. sensor networks, is an important and well established problem, though. Here also topological characteristics of the networks were taken into account (Funke, 2005). But as such networks differ significantly from road networks in many aspects results are hardly transferable.

## 3. Basics

OSM data comes in form of nodes, ways and relations. Nodes are single locations with latitude, longitude and additional tags (like the name of the location). Ways are ordered sets of nodes, describing e.g. a road or a building footprint. Relations are compositions of multiple nodes or ways, e.g. to aggregate all buildings and roads within an industrial area. Ways and relations are typically also augmented with tags that provide various information (e.g. the street or region name).

To be able to identify meaningful features for hole classification and street name extrapolation later on, we extracted all nodes and ways that describe roads in OSM and mod-

[2] http://wiki.openstreetmap.org/wiki/Quality_assurance

eled them into a directed graph $G(V, E)$ where $V$ is the set of vertices and $E \subseteq \binom{V}{2}$ is the set of edges. Additionally, we define a weight function $w : E \to \mathbb{R}^+$ which provides the Euclidean length (computed on the sphere) of each edge. For given vertices $s, t \in V$ we also define the shortest path $\pi(s, t)$ as the path from $s$ to $t$ minimizing the summed weight of the edges $\sum_{e \in \pi} w(e)$.

Moreover we break down name tags associated with ways by assigning the respective name $n(e)$ to every edge $e$ making up the way. Edges without names are labeled $n(e) =$ null. Similarly we associate with every edge $e$ a type $t(e)$ as inherited from the respective way it is part of. Here, $t(e) \in \{0, 15\}$ and reflects the hierarchy of the network (small numbers indicate important streets as motorways, while high numbers refer to living streets).

# 4. Hole Detection in Road Networks

The basic question is how to identify pairs of locations in the network where road segments are very likely to be missing inbetween. To be able to deal with the enormous amount of OSM data, we aim for methods which work without the need for manually checking large portions of the data set. Therefore, we will apply machine learning to design a hole classifier which can be used to automatically check location pair candidates.

In the following we discuss several features that are relevant for hole detection. Obviously, connectivity characteristics of the network play an important role. Therefore, we will describe thoroughly how to measure connectivity between two nodes in a road network reasonably. Finally, we sketch the complete pipeline for hole detection, including the generation of suitable training data for the machine learning approach as well as a redundancy filter.

## 4.1. Road Network Characteristics

To identify holes, we make use of several characteristics of road networks. To be specific, we are interested in the following features:

- *Connectivity.* The most important feature is how well two locations are connected via the street network. Measuring connectivity is non-trivial and therefore discussed in detail in the next section.
- *Street type difference.* Missing links between locations exhibit most likely the same street type as the mapped streets adjacent to those locations. So a hole/missing link between an interstate and a dirt road is very unlikely. Therefore we compute the minimal street type difference for two locations $v, w$ by iterating over all their adjacent edges $E(v), E(w)$ and calculating $\min_{e_1 \in E(v), e_2 \in E(w)} |t(e_1) - t(e_2)|$. If the street type is not available for one or more of these edge, we set the feature value to 0.
- *Node degree.* In typical (OSM) road networks, the average node out-degree and in-degree (i.e. number of outgoing/incoming adjacent edges) is about 2, the maximum rarely exceeds 9. Nodes with a high degree are typically important intersections and therefore have a better chance to be in a well mapped area in OSM. On the other hand, nodes with a low degree and especially dead-ends might be indicators for poor data coverage.

While street type difference and node degree can be computed quite easily, coming up with a reasonable measure for connectivity is more difficult. In the following, we design a measure based on the notion of *local stretch* that fits our purpose of hole detection well.

## 4.2. Measuring Connectivity

Intuitively, two locations in a road network that are in close proximity of each other should also have a short path within the street network. If the shortest path in the street network is much longer than the straight line distance, it might well be that parts of a street are missing.

We will first formalize this condition and provide empirical evidence for the assumption that the shortest path distance and the straight line distance are highly correlated in road networks. Subsequently, we describe common exceptions from this observation and introduce methods to deal with those.

### 4.2.1. Local Stretch Computation

Our goal is to automatically identify pairs of vertices $s, t \in V$ for which we assume that there exists a shorter path in reality than the one derived from the OSM network. We already outlined that the ratio of the shortest path distance between $s$ and $t$ and the straight line distance might be a good indicator. This ratio is called *local stretch*. In the following we refer to the length of a shortest path by $l(s, t) = |\pi(s, t)|$, and to the straight line or Euclidean distance by $d(s, t)$. Then the local stretch can be formally defined as $LS(s, t) = \frac{l(s,t)}{d(s,t)}$. As $d(s, t)$ is a lower bound for the shortest path length, the local stretch is always greater or equal to 1. The closer it is to 1 the better the connectivity between $s$ and $t$ in the road network. To compute $\pi(s, t)$ a Dijkstra run from $s$ to $t$ is the method of choice (or some accelerated variant).

In Figure 2, the correlation of straight line and shortest path distance is visualized via the local stretch value. We observe that for large shortest path distances the local stretch is remarkable small, in fact it converges to about 1.25 (so

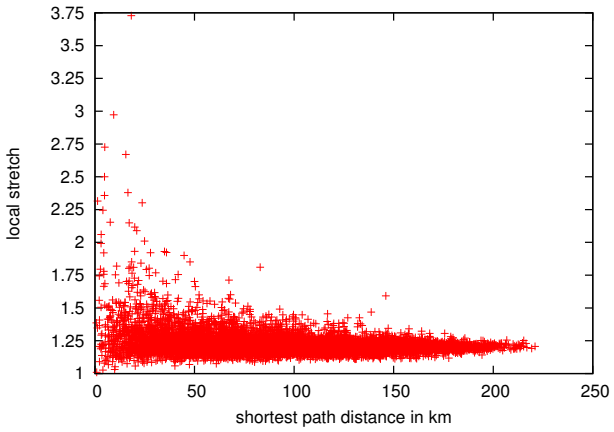


*Figure 2.* Local stretch in dependency of the shortest path length for 8,000 random point-to-point queries in Southern Germany.

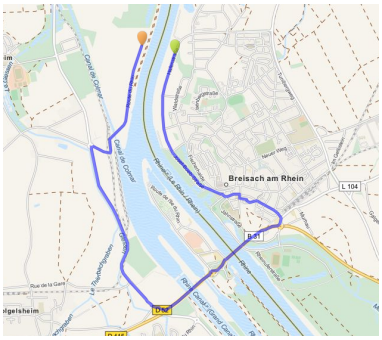


*Figure 3.* The straight line distance between the orange and the green marker is about 500 meters, but the shortest path distance is almost 8 kilometers, as the next bridge over the river is not close-by. This results in a local stretch value of 16.

the shortest path is only 25% longer than the straight line distance). For smaller shortest path distances ($< 50$ km), the local stretch values vary more and exceed 2 for some of the queries. Such point pairs with a higher local stretch than the average are good candidates for hole indication as they imply poor connectivity of the road network. Also it makes only sense to search for holes on a very local level anyhow. Holes between two far away locations are likely to be caused by (several) local holes, i.e. many missing road segments. Furthermore holes between two locations that are many kilometers apart are at least equally likely to result from poor infrastructure in the area than from missing road network data.

#### 4.2.2. Incorporating Obstacles

Unfortunately, local stretch alone is not a sufficient measure for connectivity in road networks. Consider e.g. a river which can only be crossed via few bridges, then the local stretch for two points on opposite sides of the river is high as well (see Figure 3 for an illustration). Other kinds of natural or artificial obstacles have the same effect, as e.g. lakes or interstates.

One way to overcome this problem would be to add a post-processing phase in which for each identified pair of nodes with high LS it is automatically checked whether there is some obstacle between them. But this approach imposes several problems:

- *How to decide if an obstacle blocks the hole enough.* Consider e.g. the two green points in Figure 1 (left): There is a building on the straight line between them. Nevertheless, these two points indicate a real hole.
- *Increased Runtime.* If the number of pairs is large (e.g. along every river, we expect a multitude of candidates), then to check for every single one if there is a blockage inbetween is very time-consuming even if a suitable spatial data structure for managing the obstacles is used.
- *Distorted Learning.* In the end, we want to use connectivity as a feature in our machine learning approach for hole detection. If we consider these obstacle induced high LS values in the learning process, it might affect the ability to identify real holes later on.

To overcome these problems, we introduce an approach that avoids reporting such obstacle induced high LS values in the first place. The basic idea is to incorporate obstacles already in the local stretch computation phase. Comparing the shortest path distance to the straight line distance is somewhat unfair if the straight line is blocked with obstacles. Therefore we should rather compare the shortest path between two locations in the street network to the shortest path in the plane with movement-blocking obstacles, see Figure 4 (left) for an illustration. The exact computation of the shortest path with obstacles is rather complicated and expensive (see e.g. (Mitchell, 1996)), therefore we suggest an easy way to get the approximative distance: We construct a two-dimensional grid graph covering the whole area with a cell width of e.g. 10 meter. For every obstacle, we determine all grid points that are blocked by this obstacle and remove them and all adjacent edges from the grid graph. Then a conventional Dijkstra computation in the resulting grid graph provides a feasible path, see again Figure 4 (right). We refer to the length of this path as $g(s,t)$ in the following.

On this basis, we redefine local stretch as the ratio of $l(s,t)$ and $g(s,t)$ – abbreviated by $LS'(s,t)$. Note that due to the approximative nature of our shortest path length in the plane and the fact that bridges etc. are not incorporated in this calculation, $LS'$ might be smaller than 1 (while $LS$ always is $\geq 1$). Still, the smaller $LS'$, the better the connectivity between two locations in the road network.

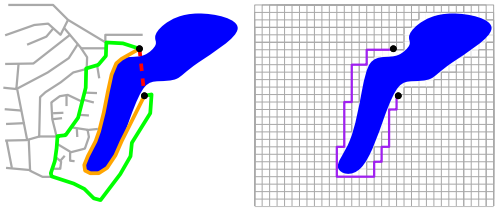

*Figure 4.* Left image: The shortest path (green) between the two black locations is much longer than the straight line distance (red). But it is not much longer than the shortest path in the plane with the lake considered as obstacle (orange). Right image: Approximative shortest path (purple) in the plane with obstacles using a grid approach.

### 4.3. Learning a Hole Classifier

With our newly designed connectivity measure $LS'$, we are now able to compute all described road network features for hole detection. As already outlined above, we are going to search for holes only between locations with a small straight line distance as otherwise we cannot hope for good accuracy – furthermore, considering every pair of locations in a large road network is computationally infeasible.

#### 4.3.1. Generating Training Data

Manual creation of a ground truth data set large enough for training the classifier is very time-consuming. Moreover, one needs to rely on the correctness/completeness of other data (e.g. GoogleMaps) for this purpose. Hence to construct a large ground truth set of classified node pairs, we used the following method: Nodes in the network that are directly connected with an edge ($LS = 1$) are no holes for sure. Also node pairs with a small local stretch do not indicate a hole with high confidence (we used 2 as a threshold in the experiments). We repeatedly selected a node in the network randomly and then searched for other nodes in close proximity with small $LS$ value. Among those we randomly picked one to form a respective pair. For each such pair we computed the feature vector and added it to the training data set. To generate training data for actual holes, we used a similar approach but removed all edges on the shortest path between the two selected nodes before computing the feature vector. In this way, we created artificial holes. For the final evaluation of the accuracy of our method, real holes will be used.

#### 4.3.2. Classifier Choice

Using the described feature vectors, the goal is to learn a good classifier which distinguishes between holes and non-holes. We expect the relationships between our features and the existence of a hole to be rather simple; for example, we expect the higher the local stretch the more likely there is a hole between two nodes. Due to these expected feature-target correlations, one suitable method for learning is *Logistic Regression*. Nevertheless, we also want to check whether there might be more complex relationships (e.g. considering node degrees). Therefore, we also used *Random Forest* as it might be able to exploit these more complex relationships.

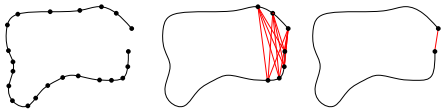

*Figure 5.* The left image shows a small cutout of a road network. In the middle image, hole candidates are indicated by red lines. The right image shows the single remaining hole after applying the extremeness check.

Both of these classifiers often work well with default parameters[3] and their learned models are fairly easily interpretable. This makes them more suitable for our task than e.g. Artificial Neural Networks.

#### 4.3.3. Extremeness Check

Feeding all reasonable node pairs into the learned classifier provides us with the set of potential holes in the network. Unfortunately, it is very likely that a single missing road segment leads to a multitude of reported candidate node pairs. If between two vertices $s, t \in V$ a segment is missing, the classifier might not only declare $s, t$ a hole but also $s', t'$ with $s'$ in close proximity of $s$ and $t'$ in close proximity of $t$. In the example in Figure 5 the problem is illustrated. This unnecessarily decreases the accuracy of our method and leads to more candidate locations that have to be manually checked in the end.

To avoid this overhead, we introduce a filter in form of an extremeness check: For every pair $s, t$ classified as hole, we inspect $LS'$ for all vertex pairs $s', t'$ with $(s, s') \in E$ and $(t', t) \in E$, i.e. all neighbors of $s$ and $t$ in $G$. If for one of those pairs $LS'(s', t')$ is larger than $LS'(s, t)$, we prune $s, t$ from the candidate list. The image in Figure 5 on the right shows the result of applying the extremeness procedure for the considered example.

The remaining candidate location pairs are then reported as the result of the automatic hole detection procedure.

## 5. Extrapolation of Missing Street Names

Another important part of the road network data in OSM are the street name tags. If a user issues a query to a route planning service, start and destination are often specified by their respective street names. This only works well if street names are complete. In OSM, though, unlabeled or only partially labeled streets are quite frequent. Often, there are multiple ways in OSM with the same street name tag but these ways are not connected (as the ways

[3] using e.g. scikit-learn (Pedregosa et al., 2011)

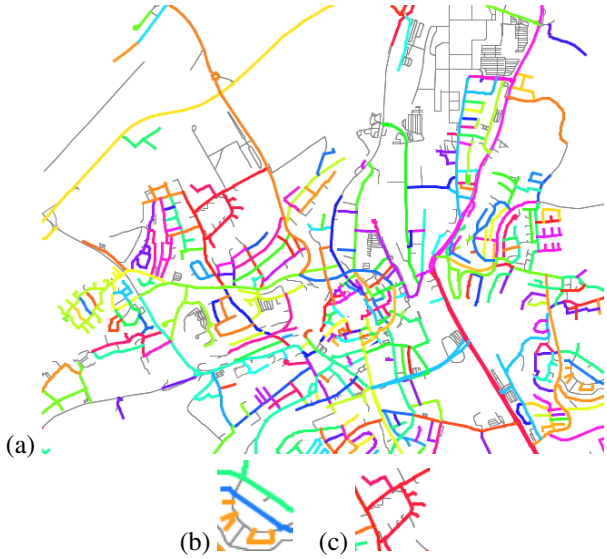


*Figure 6.* (a) Small map section based on OSM data. For every street name a random color was chosen and all segments with the same name share the same colour. Thin gray road segments are not tagged in OSM. In the second row, close-ups of (a) are shown: (b) illustrates a set of disconnected road segments with the same name (orange), and (c) shows small untagged side roads which are likely to have the same name as the red street.

might be contributed by different volunteers, but none of them mapped the complete course). If a user searches for a specific street (e.g. to see which shops are close-by), he expects a single entity to be returned and not multiple ways. Also if in a route planning query a user prefers certain streets or wants to avoid them, their names have to be fully contained in the data to account for that.

In the following we try to extrapolate missing street name tags from given data. We want to connect multiple ways with the same street name and extend partially tagged roads to completely tagged roads where possible. We describe semantic and topological characteristics of the road network that used as features in machine learning help to decide whether an untagged road segment can be labeled with high confidence.

### 5.1. Feature Extraction

We primarily rely on the assumption that all the road segments that belong to a street with one name are connected. According to our study in completely tagged areas this assumption is true for almost 99% of all streets. The visualization in Figure 6 (a) also shows typical connectivity characteristics of road segments with the same name. We refer to a connected set of edges with the same name as name component. A first feature we consider is whether an edge *is on a shortest path between two disconnected name components* with the same name (see Figure 6(b) for an example). We initially set this feature to 0 for all edges. Then we extract all name components in the network and identify street names which exhibit multiple name components in close proximity of each other (as the same street name might also occur in many villages/cities, as e.g. 'Main Street'). For every such street name we run Dijkstra computations between all pairs of nodes in different components. For all untagged edges on one of the resulting shortest paths we set the feature value to 1.

As a second feature we consider *the number of close-by name components*. So we run a Dijkstra computation from each of the two endpoints of an untagged edge until all nodes in the Dijkstra search tree are either dead-ends or are only adjacent to unrelaxed edges with $n(e) \neq$ null. For all nodes in the Dijkstra search trees we compute the set of name tags of adjacent edges. Figure 6(c) shows a small example where the feature vector entry equals 1. Being connected to a single name component might be a strong indicator for the segment to belong to this component.

But as connectivity to a single name component could also mean that only one street in the area is tagged, we also consider the *shortest path distance to the closest name component* (retrievable from the two Dijkstra runs described above) and the *number of intersections* on the shortest path from the edge to the nearest name component. The higher those two values the less likely it is that the edge belongs to that name component. Finally, we again consider the *street type*. Typically, a name component consists only of edges of the same street type. Hence the feature value is computed as the absolute street type difference of the edge and the most frequent street type in the closest name component.

### 5.2. Training Data and Machine Learning

Again, we generated a large training data set automatically. For that purpose we first extracted completely tagged streets, i.e. we searched for name components with all nodes in that component only being adjacent to tagged streets, so no surrounding street name data is missing. Then we randomly deleted less than half of the name tags from edges on this street and also from edges inside a certain radius around the street. Afterwards, we computed the feature vector for each now untagged edge on the selected street and added the result to the training data set. Furthermore we selected completely tagged streets in the same way, but removed all of its tags and some tags on edges in the neighborhood. These are examples where extrapolation is not possible. Again, we computed the feature vectors and added them to the training data.

Like for hole classification, we deem Logistic Regression and Random Forest as suitable learning methods to infer which street segments can be extrapolated.

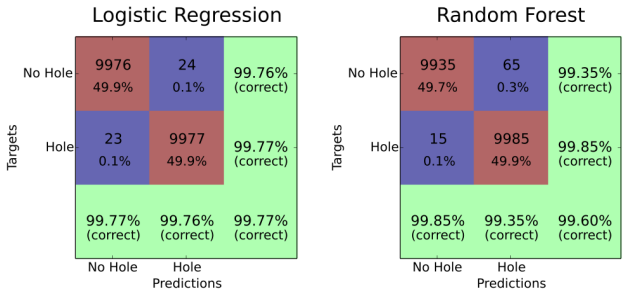


*Figure 7.* Accuracy of learned classifier on generated data using Logistic Regression or Random Forest.

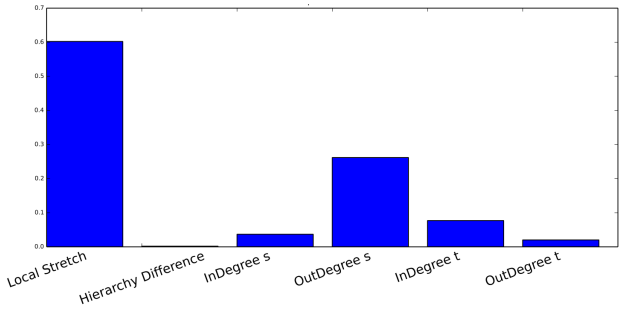


*Figure 8.* Feature importance when using Random Forest for classification.

## 6. Experimental Evaluation

We implemented the described feature extraction methods in C++. For machine learning, we used the scikit-learn package for Python (Pedregosa et al., 2011). Experiments were conducted on a single core of an Intel i5-4300U CPU with 1.90GHz and 12GB RAM. We used the OSM road network data of Germany (22.3 million nodes) and Poland (6.3 million nodes) for learning and evaluation.

### 6.1. Hole Classification

We created a data set containing 10,000 feature vectors of holes and 10,000 feature vectors of non-holes with the procedure described in Section 4.3.1 on the Germany data set.

For evaluating our machine learning pipeline on this data, we used 10-fold stratified cross-validation applying Logistic Regression and Random Forest to our training data. The outcomes are summarized in Figure 7. We observe that both Logistic Regression and Random Forest work remarkably well; both predict correctly in over 99% of the cases. While Logistic Regression achieves a better overall accuracy, Random Forest missclassifies slightly less holes as non-holes.

Having a closer look at the importance of the considered features (see Figure 8) for Random Forest, we see that local stretch is most important followed by the out-degree of $s$ and the in-degree of $t$. We also evaluated the AuC score of the features. Local stretch achieved a score of

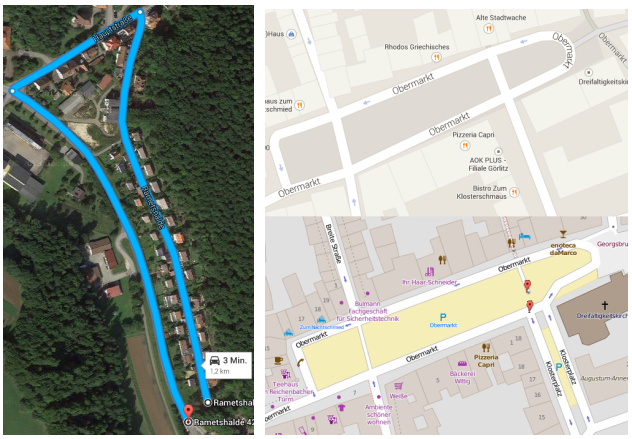


*Figure 9.* Left: Location pair falsely identified as hole by our classifier as observable when marked as start and endpoint in GoogleMaps. Right: Correctly identified hole indicated in the lower image by the two red markers on the OSM based map. The upper image shows the same cut-out on GoogleMaps with the two points being directly connected.

0.97 which underpins its importance for classification. The other features achieved AuC scores below 0.6. Nevertheless the combination of all features led to a 1-2% higher accuracy than considering only local stretch. The hierarchy difference resulting from the street types adjacent to $s, t$ does not really contribute to the classification process. One reason might be the way we generated the training data. Non-holes between e.g. parallel running motorways and federal streets which exhibit rather high local stretch but also high hierarchy difference are not likely to be included in our data set. Manually selecting such examples and adding them to the training data might increase the importance of the hierarchy feature. Another problem is that there are road segments without a type which distorts the learning process.

Finally, we used our complete pipeline for real hole detection on Germany and Poland. For evaluation, we firstly selected 2000 nodes randomly in Germany and Poland. For each such node $s$, we extracted all nodes $t$ within a radius of 500m (straight line distance) to form candidate pairs $(s, t)$. For each candidate pair we computed the respective feature vector. We extracted rivers and lakes from OSM and treated them as obstacles for the $LS'$ computation. Due to our efficient implementation of the local stretch computation, it took less than 5 minutes to process all nodes. Then we applied our classifier (Random Forest) to decide which of the candidate pairs are likely to be holes. Afterwards, we used the extremeness check to filter superfluous candidates by classifying also node pairs close to the identified holes and selecting those with the highest $LS'$ value (therefore final node pairs not necessarily include one of the randomly selected nodes in the beginning). Table 1 shows an overview of the number of pairs resulting from each step. For the re-

| | Germany | Poland |
|---|---|---|
| $d(s,t) < 500m$ | 64,194 | 44,332 |
| classified | 18,970 | 11,432 |
| extreme | 216 | 128 |
| real holes | 7 | 19 |

*Table 1.* Number of hole candidates after each step of our detection pipeline and number of correctly recognized holes in the end.

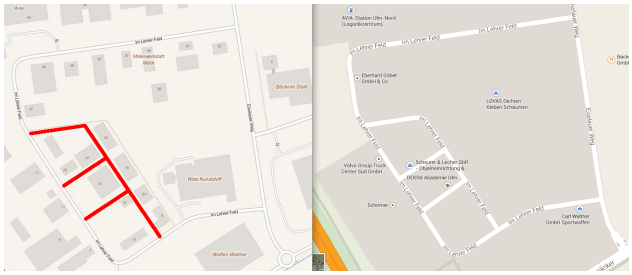

*Figure 10.* OSM based map (left) and GoogleMaps (right) on a cutout of Ulm, Germany. The red segments on the left indicate missing street names in the OSM. In GoogleMaps the correct name for all those segments is 'Im Lehrer Feld'. As the surrounding streets are tagged with this name in OSM and our approach classified the road segments as extrapolatable, the OSM data coverage can be increased here.

maining extreme candidates we checked one-by-one if they are correctly classified by comparing to satellite images and map data from GoogleMaps. Figure 9 shows examples for a falsely identified hole and a real hole. The falsely identified hole shows that it is nearly impossible to design a perfect classifier as the very same configuration of streets might very well indicate a real hole at some other location. Main sources of missclassifying non-holes as holes were clusters of one-way streets in the middle of cities, villages close to federal streets that are not directly connected and tree-like network structures in rural areas with many dead-ends. In future work, training data could be created in a way that the classifier can better deal with such scenarios.

Nevertheless, the number of pairs that have to be manually checked is significantly smaller than the number of initially created candidate pairs. The percentage of real holes among the extreme pairs is $3\%$ for Germany and about $15\%$ for Poland. The difference might result from the much better overall OSM data quality in Germany or could also be seen as an indicator for already high data coverage.

### 6.2. Street Name Extrapolation

We extracted 10,000 feature vectors for edge segments where we assume extrapolation is possible and 10,000 for edge segments where we are sure extrapolation is impossible. Then we used Logistic Regression and Random Forest to learn feature importance/weights. In our cross-validation both approaches achieved an accuracy of $99.95\%$. Also in both approaches the feature expressing whether the segment connects two name components with the same name had most influence. Despite a high AuC score, the number of close-by name components made only little difference in the learned classifiers. We assume the feature indicating the shortest path distance to the closest name component shadows the number of name components, as a very small shortest path distance and a small number of close-by components are highly correlated.

For real-world validation of our learned classifier, we selected 2000 unnamed road segments in each Germany and Poland and computed the feature vectors. Our classifier declared 235 road segments in Germany extrapolatable and 164 in Poland. A visual analysis showed that most of the segments not declared extrapolatable lied in larger untagged areas. For the road segments where the classifier indicated extrapolation might be possible, we selected the closest name component for name suggestion or, if the segment connects two components with the same name, this name is the obvious choice. We relied on a comparison to GoogleMaps and BingMaps data for evaluation. Unfortunately, in surprisingly many cases (about $10\%$) the street segments in question were unlabeled on all zoom levels or unclear or not even present in GoogleMaps or BingMaps. We excluded these cases from the evaluation. For the remaining cases, we achieved an accuracy of $96\%$ in Germany and $91\%$ in Poland (see Figure 10 for a positive example).

## 7. Conclusions and Future Work

We showed that machine learning is a useful tool to detect missing and possibly extrapolatable road network data in OSM. Making classified holes and nameless street segments with a good name suggestion available to the OSM community might raise attention to such locations and finally lead to a faster improvement of the OSM data quality.

There are various directions for future research. Our current methods do not work in regions where road data is completely missing. But e.g. mapped building footprints could be a strong indicator for the existence of infrastructure in an area. Considering buildings could also improve our classifiers. Including (large) buildings as obstacles for hole detection could lead to more realistic local stretch values in cities. Moreover, considering house numbers could significantly help to decide if a street segment should be tagged with a certain name. If the house numbers of tagged and untagged segments complement each other there is a good chance that they share the same name.

Finally, many other aspects of the OSM data might be suitable for extrapolation or classification using machine learning, e.g. distinguishing living and industrial areas or extrapolating missing house numbers.

## References


Baum, Moritz, Dibbelt, Julian, Pajor, Thomas, and Wagner, Dorothea. Energy-optimal routes for electric vehicles. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 54–63. ACM, 2013.

Delling, Daniel and Werneck, Renato F. Faster customization of road networks. In *Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings*, pp. 30–42, 2013.

Efentakis, Alexandros, Brakatsoulas, Sotiris, Grivas, Nikos, and Pfoser, Dieter. Crowdsourcing turning restrictions for openstreetmap. In *EDBT/ICDT Workshops*, pp. 355–362, 2014.

Fan, Hongchao, Zipf, Alexander, Fu, Qing, and Neis, Pascal. Quality assessment for building footprints data on openstreetmap. *International Journal of Geographical Information Science*, 28(4):700–719, 2014.

Funke, Stefan. Topological hole detection in wireless sensor networks and its applications. In *Proceedings of the 2005 joint workshop on Foundations of mobile computing*, pp. 44–53. ACM, 2005.

Funke, Stefan, Nusser, André, and Storandt, Sabine. On k-path covers and their applications. In *International Conference on Very Large Databases (VLDB)*, 2014.

Girres, Jean-François and Touya, Guillaume. Quality assessment of the french openstreetmap dataset. *Transactions in GIS*, 14(4):435–459, 2010.

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

Holone, Harald, Misund, Gunnar, and Holmstedt, Hakon. Users are doing it for themselves: Pedestrian navigation with user generated content. In *Next Generation Mobile Applications, Services and Technologies, 2007. NGMAST'07. The 2007 International Conference on*, pp. 91–99. IEEE, 2007.

Jilani, Musfira, Corcoran, Padraig, and Bertolotto, Michela. Automated quality improvement of road network in openstreetmap. In *Agile Workshop (Action and Interaction in Volunteered Geographic Information)*, 2013a.

Jilani, Musfira, Corcoran, Padraig, and Bertolotto, Michela. Multi-granular street network representation towards quality assessment of openstreetmap data. In *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '13, pp. 19:19–19:24. ACM, 2013b.

Jilani, Musfira, Corcoran, Padraig, and Bertolotto, Michela. Automated highway tag assessment of openstreetmap road networks. 2014.

Long, Ying and Liu, Xingjian. Automated identification and characterization of parcels (AICP) with openstreetmap and points of interest. *CoRR*, abs/1311.6165, 2013.

Mitchell, Joseph SB. Shortest paths among obstacles in the plane. *International Journal of Computational Geometry & Applications*, 6(03):309–332, 1996.

Mooney, Peter and Corcoran, Padraig. *Using OSM for LBS–an analysis of changes to attributes of spatial objects*. Springer, 2012.

Neis, Pascal, Goetz, Marcus, and Zipf, Alexander. Towards automatic vandalism detection in openstreetmap. *ISPRS International Journal of Geo-Information*, 1(3): 315–332, 2012.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Rahman, Kazi Mujibur, Alam, Tauhidul, and Chowdhury, Mashrur. Location based early disaster warning and evacuation system on mobile phones using openstreetmap. In *Open Systems (ICOS), 2012 IEEE Conference on*, pp. 1–6. IEEE, 2012.

Tao, Sha, Manolopoulos, Vasileios, Rodriguez, Saul, Rusu, Ana, et al. Real-time urban traffic state estimation with a-gps mobile phones as probes. *Journal of Transportation Technologies*, 2(01):22, 2012.

Vetter, Christian. Fast and exact mobile navigation with openstreetmap data. *Master's thesis, Karlsruhe Institute of Technology*, 2010.