This thesis was submitted to the
Chair of Information Systems & Databases
at RWTH Aachen University

Extending Estimation of Parking Occupancy to Untracked City Areas using City Background Information

Master's Thesis

Andrei-Eugeniu Ioniță
Matriculation Number 319298

First Examiner:	Prof. Dr. Stefan Decker Information System & Databases
Second Examiner:	Prof. Dr.-Ing. Tobias Meisen Information Management in Mechanical Engineering
First Supervisor:	Dr. Michael Cochez Fraunhofer Institute for Applied Information Technology
Second Supervisor:	André Pomp M.Sc. Information Management in Mechanical Engineering

Aachen, 04.12.17

Declaration

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Aachen, 04.12.17

Andrei-Eugeniu Ioniță

Acknowledgements

I would like to thank Prof. Dr. Stefan Decker for being the main examiner and supporting this thesis. Furthermore, I want to thank Prof. Dr.-Ing. Tobias Meisen for being the second examiner on behalf of the Institute of Information Management in Mechanical Engineering.

I also want to express my gratitude to my supervisors, Dr. Michael Cochez and André Pomp M.Sc., for the their numerous suggestions, constructive feedback and the continuous support throughout this thesis.

My current employer, Knowledge Exchange AG, I would like to thank for being flexible regarding my working hours and granting me time off to finish this work.

Last but not least, I would like to thank the S O NAH start-up team, Christian Bartsch, Thomas Grimm, and Victor ter Smitten that got me started on this project in the first place.

Abstract

Several smart cities around the world have begun monitoring parking areas in order to estimate free spots and help drivers that are looking for parking. The current results are indeed promising, however this approach is limited by the high cost of sensors that need to be installed throughout the city in order to achieve an accurate estimation rate. This work investigates the extension of estimating parking information from areas equipped with sensors to areas that are missing them. To this end, similarity values between city neighborhoods will be computed based on background data, i.e., from geographic information systems. Using the derived similarity values, the adaptation of occupancy rates from monitored- to unmonitored parking areas will be analyzed.

Contents

1. Introduction

1.1. Motivation

Parking is a known problem in cities. Worldwide we are experiencing an increase in the number of cars [21]. Currently, about 30% of the traffic in cities is caused by cars that are actively searching for parking [12]. Drivers often end up double-parking their cars, which blocks other cars, thus causing unneeded stress. Drivers that are circling for a parking space may cause safety issues, as they are often distracted and are not paying attention to cyclists and pedestrians. When circling for parking spaces, additional fuel is consumed, which, moreover, affects the environment as a result.

The solution to the parking problem is leveraging the advanced technology that we have access to. Existing parking spaces can be more efficiently occupied if the drivers know about their availability. Driving in congested car parks or busy streets to find out whether parking is available is not an option. The information about existing parking spaces needs to be available in advance, so that drivers can take the decision to drive towards a highly probable free parking space early enough and not get stuck in a traffic bottleneck.

A system for this purpose would, ideally, take into account factors such as current parking space availability, traffic, events, and weather. The driver would then be able to choose the driving routes and plan in advance accordingly. They might need to take an umbrella with them, if the found parking space is too far away from their actual destination and outside it is raining. If need it be, they may even forget about the car and take the public transportation to arrive on time and without hassle at the destination.

Obtaining the piece of information about future free parking spaces arguably requires a complex system and the approaches to achieve it vary greatly. Many such forecasting systems start by collecting statistics on the free parking spaces. Usually, it is mounted sensors that observe when a car occupies and leaves a parking space.

Various statistics draw on parking information collected over a reasonable amount of time, such as, for example, the probability that a parking space is free at a certain time of day, that on weekends the location is less occupied than on weekdays, that on rainy days there are more parking spaces occupied, but the parking duration and the waiting time is shorter too. Or that, when a concert is taking place not far away from the parking location, virtually all parking spaces get occupied during the respective time period.

Forecasting systems can hence derive various facts that prove useful to drivers searching for a parking space. Arriving at statistical conclusions about parking is relatively easy, when we know, what we are after. Acquiring the required data, however, be it on the parking spaces themselves or the complementary information, is, most of the time, the bottleneck for these approaches.

There have been many attempts that dealt with the scarcity of parking data with reasonable success. For car parks, there are numerous systems that hold an overview at any time of the individual parking space occupancy. These systems provide real-time parking monitoring and are able to guide a car to a precise parking space, once it has entered the car park. On the other hand, car parks are rather controlled environments, exclusively dedicated to parking and whose optimization is often economically motivated by commercial affiliations. Keeping track of occupancy and making predictions in this case is relatively straightforward.

Roadside parking is, in comparison, a jungle when it comes to parking space prediction. There are various hazards that occur here: not all spaces on the street are for parking, parking places are often used by stopped vehicles that load/unload, etc. Moreover, their occupancy is not necessarily motivated by shopping and there is no economic incentive for the managing municipality to monitor them. It is understandable that municipalities do not want to go to the lengths of monitoring all roadside parking spaces with sensors. The on-street parking problem continues to be a challenge, for which currently there is no standard solution.

1.2. Approach

The current approach is motivated by the insufficiencies of on-street parking prediction systems. It sets out to address the shortage of available parking data and the hard-to-anticipate parking situation on city streets.

This work will follow up on similar approaches in the area of smart parking. We will look at existing systems that have gathered on-street parking data for certain cities. Additionally, we will make use of additional city data, obtained from sources not related to parking. We aim to offer answers to the following questions:

1. Are parking occupancy rate predictions more precise when realized on small city areas rather than for the whole city?

2. Can parking profiles of city areas be created and quantified by taking into account city background data?

3. Can a parking prediction system based on parking data from a city area be transferred to another city area *without* parking data by taking into account the "parking profiles" of the respective city areas?

4. Can the processing of the collected data for such an approach be standardized, so that it is extensible and generally applicable to parking in cities?

To realize this approach and verify the hypotheses, we need access to relevant data. Our requirements for this system are:

1. *on-street parking data*, preferably collected over multiple months

2. *additional city data relevant to parking*, such as types of services and their localization

1.3. Contribution

Despite the relative large number of systems that already exist, which are tackling the parking problem, the approach presented in this work is original and goes about the problem differently, to the best knowledge of the author. First of all, it sets out to compute parking occupancy rates for locations where no parking data has been recorded. Secondly, it uses data from an independent source to the parking data itself to compute the parking predictions. Thirdly, it proposes the concept of "parking demand profile" and quantifies it, before applying it to calculate parking predictions. Fourthly, it deals with semantically annotated information for all its data sources.

1.4. Thesis Organization

The thesis is structured as follows:

1. **Introduction** (cf. Chapter 1) states the motivation for the problem and formulates the research questions.

2. **Problem Definition** (cf. Chapter 2) elaborates the goals set in the *introduction* by means of examples and bridges the understanding path towards the rest of the work.

3. **Background** (cf. Chapter 3) introduces the reader to the concepts that are needed for understanding the proposed approach.

4. **Related Work** (cf. Chapter 4) presents similar contributions to the field of smart parking and compiles an inventory of their characteristics.

5. **Design & Implementation** (cf. Chapter 5) presents the proposed solution in-depth by including theoretical and implementation details.

6. **Evaluation** (cf. Chapter 6) carries out the instantiation of the proposed approach with a concrete use case, presents the results and analyzes them.

7. **Conclusion & Future Work** (cf. Chapter 7) summarizes the outcome of the approach and points out aspects that could be further pursued.

8. **Appendix** (Appendix A) contains supplementary evaluation information that otherwise does not fit in the respective chapter.

2. Problem Defintion

This section will help to better understand the problem we are solving. It will start by addressing the main goals formulated in Section 1.2 and derive further subgoals. The subgoals discussion acts as a bridge towards the rest of the work.

2.1. Parking Profiles

Building parking profiles for city areas is one of the goals set in Chapter 1. Such a parking profile is reflected by the parking demands throughout the city. Since we are looking for a way to quantify parking, we will look for indicators that attract or assume parking.

Office buildings areas, for instance, are places where the parking demand is high. Dedicated car parks, usually, do not manage to provide enough supply of parking, so many drivers end up leaving their cars on the side of the street. For office building areas, this happens during working hours, generally between 8 – 18. If there would be a way to localize the office buildings, we could conclude that two such areas, perhaps in different parts of a city, or even in two different cities, have a similar parking situation, i.e., that during 8 – 18 on weekdays there is a high demand in parking for both. In case of restaurants, on the other hand, we see a spike in parking demand in the evening, usually from 18 – 22, and more so on weekends. In residential areas, the cars are parked in the evening and leave again early morning. A measure that would capture the parking demand will therefore be based on the stay duration of customers or employees of the particular services.

Getting access to data about establishments and services in the city would therefore enable us to quantify the parking demand. We shall call this type of data *city data* or *city background data*.

2.2. Transferring Parking Models

By parking models we understand machine learning systems that statistically forecast parking occupancy. In order to build such models, data on parking occupancy is needed. Also, data about relevant circumstances such as parking price, traffic, events, and weather is beneficial. We shall call the collection of both these data types *parking data*.

Instead of creating a machine learning model for the whole city area that has parking data, there are chances to achieve better approximations when considering models for smaller areas. Smaller areas are more specialized and produce models that are transferable to other city areas.

Once parking models are created, we will define how to apply them on areas different that their original ones. Areas that do no have parking data are not suitable for building such a prediction model. These areas will need to "import' a model from another area that *has* parking data. The two areas might be different in terms of parking demand, so applying an imported model would only make sense if the difference in parking profiles is taken into account.

Summarizing, for transferring parking models the following elements are needed:

1. a city area A that has parking data, with which a parking model is built
2. a city area B that has no parking data
3. both city areas have a parking profile created based on their corresponding parking demands
4. applying the parking model from A to B will need to take the areas' parking profiles into consideration

Note that, in order to obtain good results from model transferring, the city and the parking data need to be independent. If the two sources would have elements in common, then the application of model A to area B would be biased.

Finally, we will aim to make our solution extensible and generally applicable to parking in cities. In order to establish a common format, all the processed data will be considered in RDF format, which will make it more easy to distribute and process.

3. Background

This chapter introduces some prerequisite theoretical foundations. It also includes concepts and systems that are being referenced throughout the rest of this work.

3.1. Resource Description Framework

The Resource Description Framework (RDF) [48] [47] [49] expresses semantical information about resources. A *resource* is an abstract concept, it can be a document, a person, an object, etc. RDF is a format designed to be used between applications, yet there are tools that make RDF human-friendly. RDF acts as a standard format for exchanged information, so that the sender and recipient align when formatting and parsing the data.

RDF uses the Web to publish information. A resource has an identifier to which various other resources link to. The identifiers are called International Resource Identifiers (IRIs) and can be thought of as abstract Universal Resource Locators (URLs). The links between resources are called *properties*. Resources can also link to basic information called *literals*, i.e., strings, numbers, dates, or booleans. A collection of data about resources that are interlinked together is referred to as Linked Data [40].

RDF is used for a number of reasons. It provides mark-up information for search engines and makes indexing websites easier. RDF also builds a vocabulary of data for certain domains, hence acting as a compendium and providing access to comprehensive information about the respective domains for applications that want to use them. RDF also makes for a data interface between communication partners. For more practical uses of RDF, see the official documentation linked above.

The building block of RDF is the *statement*, also a called a triple. A statement tells something about a resource. It contains a subject, a predicate, and an object. The subject is a resource, the predicate is a property, while the object can be a resource or a literal. A collection of triples may state information about a resource several times (cf. Listing 3.1) and the entire linked data can be visualized as a graph (cf. Figure 3.1). Given such a collection of statements and the fact that RDF documents are not necessarily human-readable, one can extract the information they are looking for by using SPARQL [50]. SPARQL is a RDF query language that resembles SQL in concept. Its syntax is somewhat different from standard SQL but it can be learned relatively easily when knowing the latter.

Listing 3.1: A simple collection of RDF statements written in informal syntax

```
<Max> <is a> <Student>.
<Max> <writes> <Paper>.
<Paper> <is due on> <01-03-2018>.
<Max> <likes> <hiking>.
<hiking> <is activity?> <true>.
```

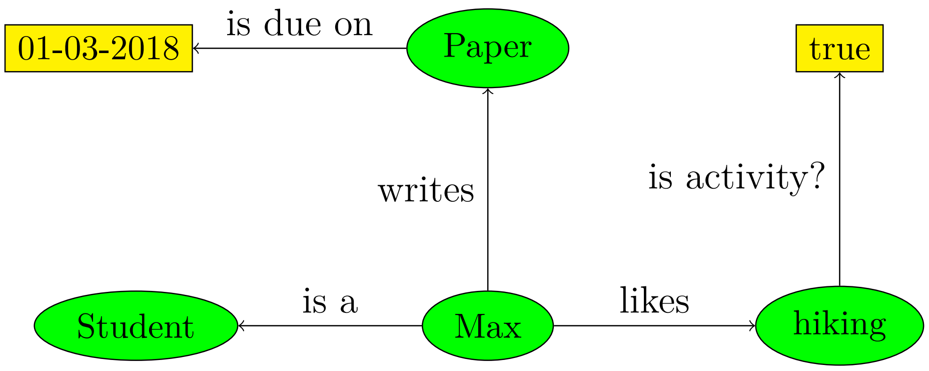


Figure 3.1.: The graph visualization of the statements in Listing 3.1

RDF statements about objects in the world need to comply to an RDF schema in order to be properly understood by the party they are communicated to. An RDF schema defines a vocabulary. Such a vocabulary can be easily understood by following the analogy to the word dictionary, which imposes type restrictions on words. There are only certain IRIs (identified words) that can be used as resources (nouns, pronouns), as there can be only some IRIs that are valid as properties (verbs), as there can be only be some literals that qualify as nouns or numerals.

The very first, and still most well-known RDF vocabularies are Dublin Core[1], schema.org (the vocabulary for search engine mark-up)[2], SKOS (vocabulary on thesaurus information)[3]. RDF vocabularies are meant to be reused and to become a lingua franca for information exchange in the field they represent. Large collections of RDF data being used today are Wikidata[4], a collaborative and multilingual platform run by the Wikimedia foundation, DBpedia[5], WordNet[6].

[1]Dublin Core: `http://dublincore.org/documents/dcmi-terms/`
[2]schema.org: `http://schema.org/`
[3]SKOS: `https://www.w3.org/2004/02/skos/`
[4]Wikidata: `https://www.wikidata.org/wiki/Wikidata:Main_Page`
[5]DBpedia: `http://wiki.dbpedia.org/`
[6]WordNet: `https://www.w3.org/2006/03/wn/wn20/`

In practice, RDF appears under different formats. N-Triples, Turtle, N-Quads have specific syntaxes and can be used to exchange large sets of RDF information, while not intended to be human-readable. In other formats, such as JSON-LD, RDFa, or RDF/XML, the statements appear encoded in JSON or XML syntax respectively.

As a consequence of its statement-centered definition, RDF statements can also be seen as mathematical logic statements. Hence, logical reasoning can be applied on them and new information can be inferred about resources in a given collection. Some entailments can however lead to inconsistencies.

An implementation of the RDF standard is provided by Apache Jena [1]. Its Java API enables developers to create RDF models and extract information using SPARQL queries.

3.2. Machine Learning

Machine Learning can be loosely defined as a collection of techniques through which an application learns from "experience" and without having to be programmed to do so. The term was coined in 1959 by Arthur Samuel, an IBM computer gaming pioneer [17]. Machine learning applications can be categorized into supervised and unsupervised.

In supervised learning, applications are first presented with inputs for variables called *features*, alongside the desired outputs, whose variables are known as *target variables*. The phase in which the application processes this data is called *training*. Afterwards, the applications are asked to compute outputs for unseen inputs, a phase called *testing*. This is realized by determining rules in the data that map inputs to outputs and build a *model* from them. In unsupervised learning, applications are only presented with inputs for which a structure needs to be found. Examples of unsupervised learning methods are clustering and dimensionality reduction.

Supervised applications are of two types: classification and regression. Classifiers are models that have discrete outputs, i.e., labels. A classifier will therefore typically assign one out of multiple labels to future inputs, after having been trained with a collection of inputs and their corresponding labels. A common example of classifiers are spam filters, where the applications are trained to categorize e-mails into spam or non-spam, based on its subject, sender and content.

Regression applications are models for which the output is a real number. Such models are therefore typically represented in Cartesian coordinates. An example is linear regression, where the model estimates y-axis values that are a linear combination of its x-axis counterparts.

In the present work, we will deal with supervised methods when training parking occupancy estimation models, but also with unsupervised models when clustering the parking location units.

3.2.1. Decision Trees

Decision trees is a form of classification model that learns to determine the target variable by modeling it as a decisional process. The process resembles a connected acyclic graph, i.e., tree, where decisions are made at the vertices and the edges are the decisional paths. The classification of a new instance begins at the root and drops down the tree, ending in a leaf. Each vertex represents subsets of the feature value domains which are established in the trained phase. The values of the leafs are the target variables. For an example of decision tree, see Figure 3.2.

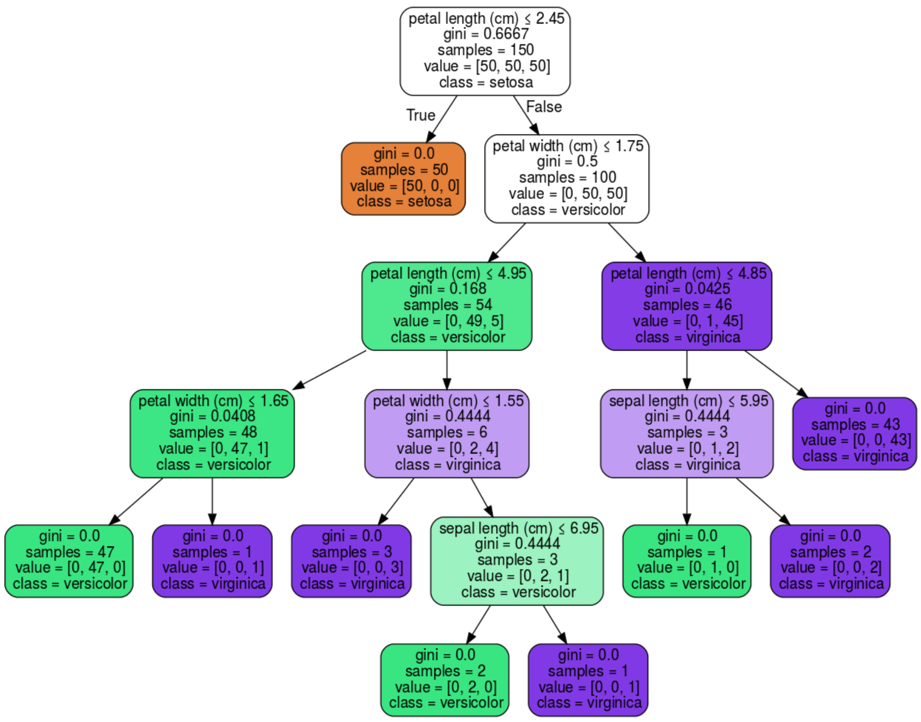

petal length (cm) ≤ 2.45 gini = 0.6667 samples = 150 value = [50, 50, 50] class = setosa
True
False
gini = 0.0 samples = 50 value = [50, 0, 0] class = setosa
petal width (cm) ≤ 1.75 gini = 0.5 samples = 100 value = [0, 50, 50] class = versicolor
petal length (cm) ≤ 4.95 gini = 0.168 samples = 54 value = [0, 49, 5] class = versicolor
petal length (cm) ≤ 4.85 gini = 0.0425 samples = 46 value = [0, 1, 45] class = virginica
petal width (cm) ≤ 1.65 gini = 0.0408 samples = 48 value = [0, 47, 1] class = versicolor
petal width (cm) ≤ 1.55 gini = 0.4444 samples = 6 value = [0, 2, 4] class = virginica
sepal length (cm) ≤ 5.95 gini = 0.4444 samples = 3 value = [0, 1, 2] class = virginica
gini = 0.0 samples = 43 value = [0, 0, 43] class = virginica
gini = 0.0 samples = 47 value = [0, 47, 0] class = versicolor
gini = 0.0 samples = 1 value = [0, 0, 1] class = virginica
gini = 0.0 samples = 3 value = [0, 0, 3] class = virginica
sepal length (cm) ≤ 6.95 gini = 0.4444 samples = 3 value = [0, 2, 1] class = versicolor
gini = 0.0 samples = 1 value = [0, 1, 0] class = versicolor
gini = 0.0 samples = 2 value = [0, 0, 2] class = virginica
gini = 0.0 samples = 2 value = [0, 2, 0] class = versicolor
gini = 0.0 samples = 1 value = [0, 0, 1] class = virginica

Figure 3.2.: An example of a decision tree applied on the famous Iris dataset [8]

3.2.2. Support Vector Machines

Support vector machines are a class of machine-learning algorithms used in classification and regression problems. They are applicable both in linear and non-linear settings. The idea behind the main algorithm is finding boundaries between points belonging to different classes, so that the nearest points to the boundary are as far as possible from each other. More rigorously, the algorithm finds so-called maximum-margin hyperplanes in the multidimensional space of points. The hyperplanes cannot be further apart from either class of points. The instances that are closest to a maximum-margin hyperplane are called *support vectors*. Each class has at least one support vector. The support vectors uniquely define a maximum-margin hyperplane and all other instances are hence redundant in this respect. See Figure 3.3 for an example of SVM line separation in the bidimensional case.

Support vector machines are often used to train non-linear models. By replacing the dot products of the vectors with a higher-degree kernel function, the instances are mapped into a higher dimensional space. A hyperplane that is furthest away from the class points it separates assigns different classes to the respective instances. Common kernel function are the polynomial kernel, the Gaussian radial basis function, and the hyperbolic tangent.

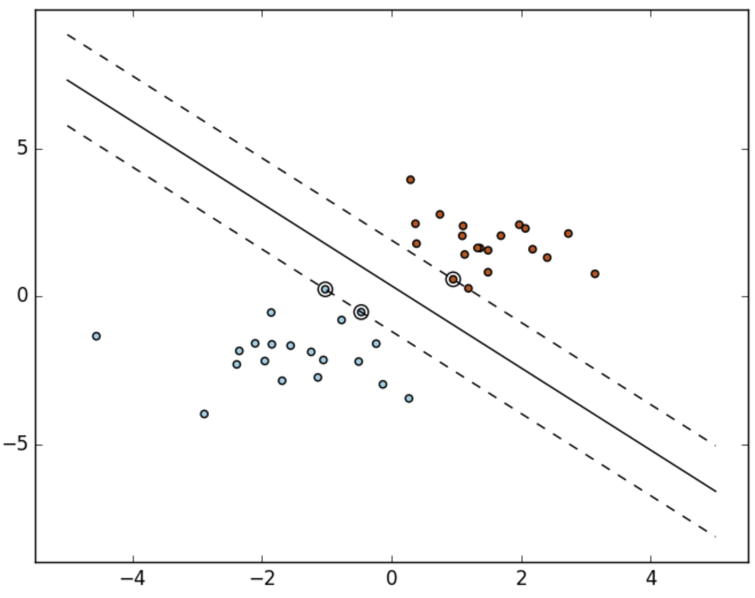


Figure 3.3.: SVM hyperplane separates two classes of points in 2D. Circled are the support vectors [38]

3.2.3. Multilayer Perceptrons

Multilayer perceptrons are graph structures that process data and are organized as oriented graphs. The edges are typically represented from left to right and the vertices are vertically aligned as in layers (cf. Figure 3.4). Each node is a perceptron, a structure that is activated by some specific input and that fires an output. The inputs are the weights of the incoming edges, while the outputs are weights of the corresponding outgoing edges. The leftmost nodes are the input nodes. These contain the feature values of the model. Additionally, there is a bias node among the input nodes. The right-most node is the output, it contains the target value. The in-between layers are called hidden layers and their network is only controllable though model meta-parameters. The decision based on which the perceptrons fire is typically derived from the features of the model. However, the decisional processes of the multilayer perceptrons are often viewed as black-boxes.

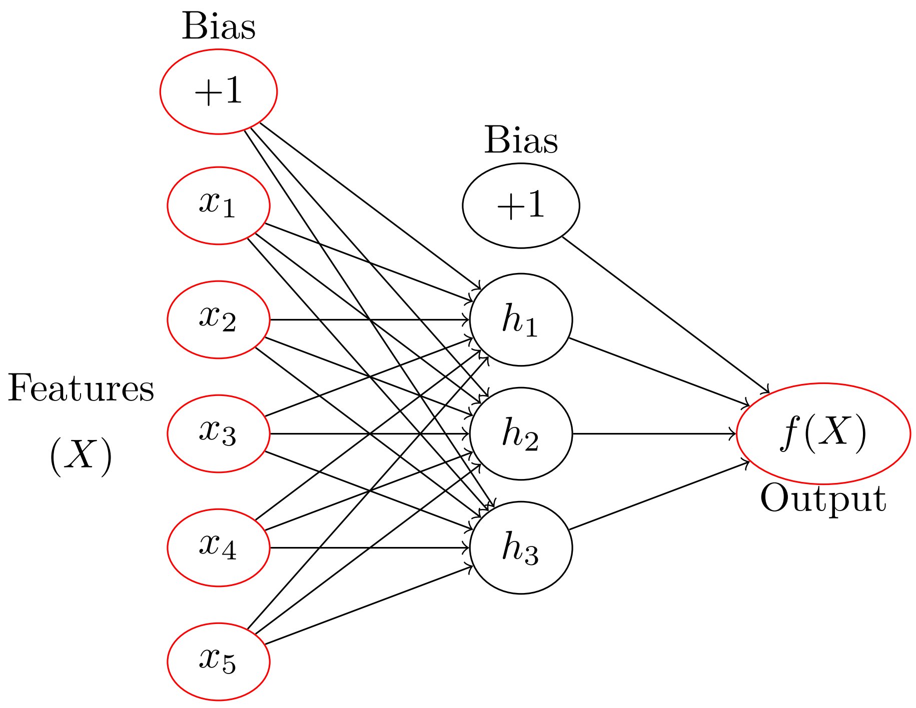


Figure 3.4.: An example of MLP with one hidden layer

3.2.4. Gradient Boosted Trees

Gradient boosted trees are a fairly recently developed regression method [44]. The algorithm relies on two parts: boosted trees and the gradient descent algorithm.

When training a decision tree model to minimize a loss function, residuals are inevitably resulting, i.e., absolute differences between the model and the target values. However, the residuals can be seen as training data for a new model, i.e., another decision tree, that would, in turn, try to approximate the new training values. Nevertheless, a new round of residuals emerge. Continuing the process, further decision tree models will try to fill in the gaps. The sum of all the models will, in the meantime, approach the original goal function asymptotically. The approach is called *tree boosting* and an example can be seen in Figure 3.5.

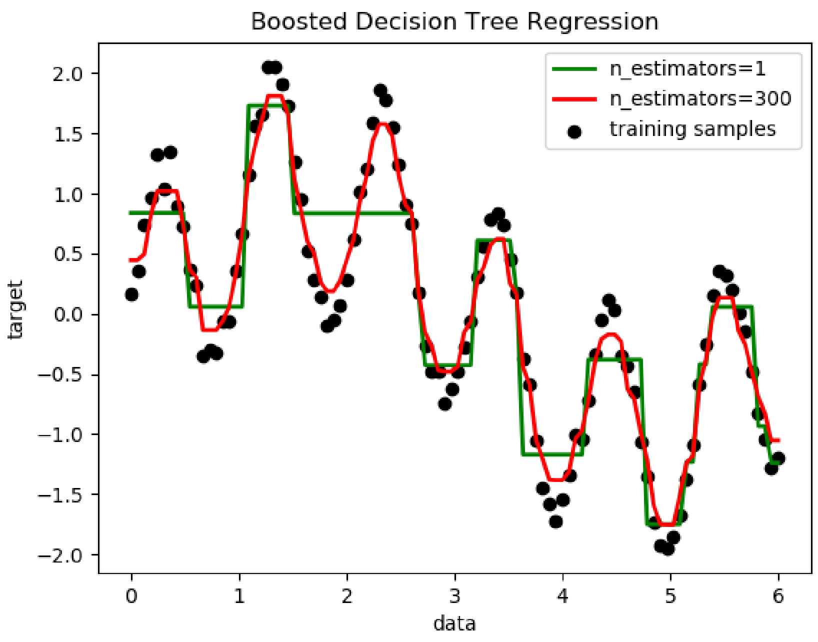


Figure 3.5.: Boosted trees regression approximates a sinusoidal dataset [7]

The approach above has something in common with the *gradient descent algorithm*. The gradient descent optimization algorithm attempts to find the minimum of a given function. It proceeds by taking steps towards the steepest slope of the function in order to move closer to the minimum. After some iterations, it will eventually reach a local minimum.

To see the similarity to the boosted trees approach, consider the residuals from the tree boosting approach above as negative gradients in the gradient descent algorithm. The solving of the initial problem would then boil down to applying a gradient descent strategy, while iteratively producing decision trees until a local optimum is reached and the overall resulting model cannot be improved any longer.

3.2.5. Cross-Validation

Cross-validation is technique for evaluating the performance of a predictive model [5]. It is used to assess a model before it has been applied on a test dataset. Cross-validation starts by splitting the original training set into P subsets. In one round of cross-validation, the model is trained on the data from $P-1$ subsets and then tested against the target output of the remaining subset. In order to reduce variability, multiple rounds of cross-validation are performed and the result is averaged.

3.2.6. Scoring Functions

Model evaluation techniques such as cross-validation may use various metrics to arrive at a performance measure of a model. Some of the scoring functions are designed for classifiers, such as accuracy and precision-recall.

As this work deals with regressions models, we are interested in regression metrics. The coefficient of determination, R^2, establishes the degree in which the model can correctly predict future values [28]:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n_{samples}-1}(y_i - \hat{y})^2}{\sum_{i=1}^{n_{samples}-1}(y_i - \bar{y})^2}. \tag{3.1}$$

where y and $\hat{y}$ are the vectors of real and, respectively, predicted values.

The *mean absolute error* (MAE) computes the expected value of the absolute error loss [18]

$$MAE(y, \bar{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} |y_i - \bar{y}_i|, \tag{3.2}$$

where $\bar{y}$ is the mean of the y value.

The *root mean squared error* (RMSE), similar to the mean squared error, computes the expected value of the squared error [19]

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2} \tag{3.3}$$

We shall use RMSE later on for model testing, as it generally more intuitive than R^2 [20].

3.2.7. Grid Search

Training a machine learning model in practice requires some tuning. Model training is conditioned to specific hyper-parameters, depending on the model, that determine its learning. Certain combinations of hyper-parameters yield better models, i.e., with a lower training error. However, since tuning hyper-parameters requires an in-depth knowledge of the theory behind every model, we will use *grid search* for training. The input for grid search are value sets for the hyper-parameters, which are subsets of the value domains of the hyper-parameters. On one iteration, grid search will take a sample value for each hyper-parameter and assign it to the model. A variant of grid search runs up to the point that models are trained with the Cartesian product of the hyper-parameters, provided the value sets are discrete. A randomized version of grid search runs by generating a specified number of models by randomly selecting values from the hyper-parameter values sets.

In Python's *scikit-learn* package, the grid search functions are called GridSearchCV and RandomizedSearchCV, whose behavior matches the ones described above.

3.3. Cosine Similarity

The cosine similarity between two vectors is defined as the cosine of the angle between the two vectors. It therefore varies between -1 and 1. The larger it is, the less is the angle between the two vectors. Expressed mathematically, the cosine similarity $cos(\theta)$ between two vectors A and B is [4]:

$$cos(\theta) = \frac{A \cdot B}{\|A\|_2 \|B\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3.4}$$

where A_i and B_i are the components of vector A, respective B. The power of the cosine similarity lies in wide-range of vector representations, to which many concepts can be reduced.

3.4. Earth Mover's Distance

The earth mover’s distance (EMD) is a measure used in statistics that roughly expresses the difference between position and magnitude of two curves. It is best explained by regarding the curves as the hull of earth piles. For two separate earth piles, EMD computes the minimum effort of rearranging a pile so that the shape of the other pile is obtained. Moving P particles over a distance D is equal to the effort $P \times D$. A prerequisite for this operation is that the two piles need to contain the same quantity of earth.

More rigorously, the earth mover's distance is better known in mathematics under the name Wasserstein Metric. Given two normal distributions $\mu_1 = \mathcal{N}(m_1, C_1)$ and $\mu_2 = \mathcal{N}(m_2, C_2)$, where m_1 and $m_2 \in \mathbb{R}^n$ are their respective expected values and C_1 and $C_2 \in \mathbb{R}^{n \times n}$. Then, the 2-Wasserstein distance between μ_1 and μ_2 is [39]:

$$W_2(\mu_1, \mu_2)^2 = \|m_1 - m_2\|_2^2 + trace(C_1 + C_2 - 2(C_2^{1/2} C_1 C_2^{1/2})^{1/2}) \quad (3.5)$$

In practice, we will not apply the Wasserstein metric directly in practice, but rather resort to some levels of discretization. First off, a number of so-called bins is determined. Each bin represents a unit on the X axis, the same on which the visiting duration is expressed. We will take a number of buckets equal to the maximum amenity mean and add $3\times$ the largest standard deviation to it, as it is known that within $3\times$ standard deviation on both sides of the mean over 99% of the Gaussian sum is covered. Moreover, an offset on the X-axis equal to $3\times$ the maximum standard deviation is used. This way, we are sure the landscape of summed Gaussians will easily fit into the number of bins.

3.5. Clustering

Clustering are data mining techniques that seek to group data instances together based on some natural ordering. The natural ordering can vary and is specific to the problem to be solved. Here, a clustering method called *k-means* will be briefly described, as it makes the subject of its application later on.

The domain of application of k-means is numeric, i.e., the data instances need to have a (multidimensional) numeric representation. A prerequisite for the application of *k-means* is the number of final clusters that are sought, which we will denote by k. The method starts by initially selecting k points at random from the data points. The k points are the initial cluster centers. The points that are nearest to a cluster center are assigned to that cluster. The distance metric is considered the Euclidean distance. Next, for the formed clusters, the centroids are calculated based on the containing points. The cluster centers are re-assigned to the calculated centroids and the process is repeated. The process stops when the points do not change their assigned cluster for two iterations in a row.

The advantages of k-means lie in the fact that it is simple and effective. Each cluster minimizes the sum of squared distances from its points to the centroid. The entire point space is partitioned in a so-called Voronoi diagram. However, the distance minimum is a local minimum. This means, there may exist a clustering for which the total sum of distances for all clusters is smaller. It is the initial selection of seeds, i.e., cluster centers, that has an impact on the resulting clusters.

To overcome the suboptimal result, one can either apply the algorithm multiple times and then choose the best result, i.e., the one that has the minimal total squared distance. Or, one uses an improved version of k-means, *k-means++*. This chooses the first seed with a random uniform probability and the second seed with a probability proportional to the square of the distance from the first seed. All further seeds are chosen with a probability proportional to the distance to the nearest already chosen seed.

An application example of the algorithm yields the results in Figure 3.6.

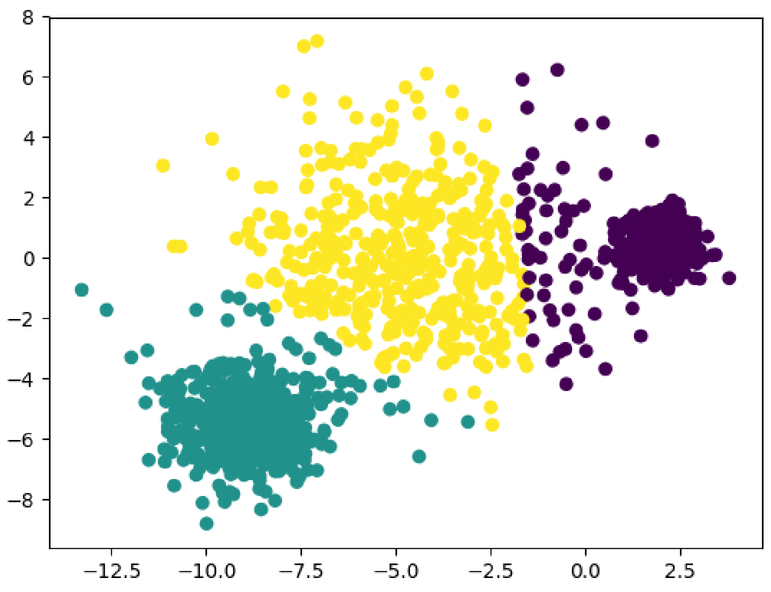


Figure 3.6.: An application of k-means with 3 clusters in Python using the scikit-learn package [9]

3.6. Correlation Coefficients

In the evaluation chapter, we make use of correlation measures to analyze the results. In statistics, correlation is a relationship between two sets of variables X and Y that establishes whether there is a linear relationship between X and Y [3]. The Pearson coefficient [27] takes into account the absolute values of X and Y when calculating correlation:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3.6}$$

where x_i and y_i are single samples of X and Y respectively, while $\bar{x}$ and $\bar{y}$ are X and Y's means. The resulting value ranges between -1 and $+1$. The end points are achieved when X are perfect monotone functions of Y, i.e. either X correlates positively with Y, when the function is ascending, or X correlates negatively with Y, when the function is descending. A coefficient of 0 indicates no correlation at all.

The Spearman's rank coefficient [35] takes into account only the ranks of the samples in X and Y when calculating the correlation. A rank is the assignment of a label to a value within a set according to an ordering relation, i.e., “first”, “second”, “third”, etc. [29]. Spearman's rank coefficient is equal to the Pearson coefficient for the ranks in X and Y.

3.7. Coordinate Reference System

A Coordinate Reference System (CRS) is a projection of the globe surface, or parts of it, onto a two-dimensional map surface. CRSs are defined by the Open Geospatial Consortium (OGC) and are represented in well-known text (WKT) format. CRSs are referenced by a Spatial Reference System Identifier (SRID) or EPSG codes defined by the International Association of Oil and Gas producers [34]. CRSs typically differ in the type of projection, the area covered and their units of measure. Spatial operations between them can be performed by first using coordinate system transformations. We shall work with EPSG 4326, which expresses points in latitude and longitude and has the degree as its unit of measure [10].

3.8. OpenStreetMap

OpenStreetMap (OSM) is a collaborative mapping service [23]. Users can annotate various locations and add informational metadata, e.g., one can mark any shop, the fact that it sells drinks and snacks, and its daily schedule. Or, a building can be annotated as hosting a restaurant, its specific cuisine, price range, whether a reservation is needed, and when it is open. Just about any public amenity is marked on the map. This data is being saved by OSM in layers. There are points-, lines-, and polygons layers. Points of interest (POIs) contain solely metadata concentrated in points on the map (cf. Figure 3.7), lines stand for routes and street-related content and polygons describe buildings and areas such as, e.g., university campuses or parks. POIs represent public amenities, various services, conveniences that are of general interest. The amenity information will be used as city data in our approach.

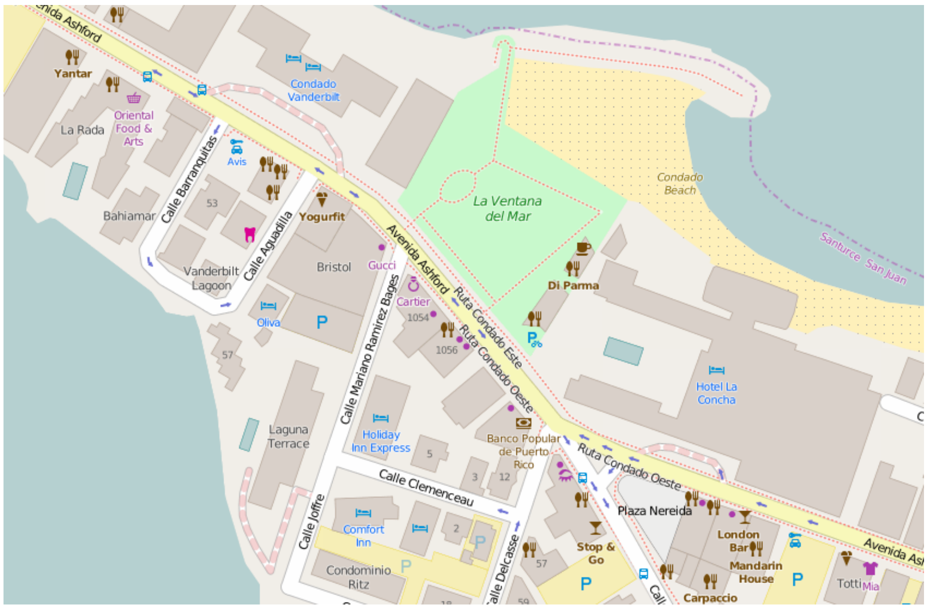
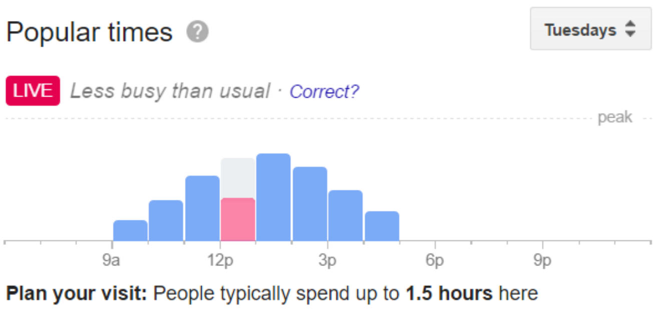


Figure 3.7.: Points of interest in OpenStreetMap [24]

3.9. Google Visit Duration

Google Places determines the visit length of customers while they are located in various businesses in the city [13]. The information is displayed in Google Places and is based on the mobile phone localization feature. This information is crowdsourced from people that have entered the same shops, restaurants, etc. and based on their length of stay, the average visit duration was derived. Usually, the minimum and maximum time of stay are provided, sometimes only the latter (cf. Figure 3.8). This Google feature is not available in every location and is offered only in places that have received enough crowd data. We shall use the Google visit duration to complement the amenity information provided in the OSM POIs.



Plan your visit: People typically spend up to **1.5 hours** here

Figure 3.8.: Example of visit duration information displayed in Google Places [14]

3.10. Leaflet

Leaflet is a JavaScript library optimized for displaying interactive maps [16]. Its clearly structured API and easy-to-follow tutorials make it a recommended library among GIS developers. We will use Leaflet to display our results and highlight city areas.

3.11. The SF*park* Project

The SF*park* project was undertaken by the San Francisco Municipal Transporation Agency (SFMTA), the city agency that manages the city's transportation, which includes on-street parking [31] [33]. The SFMTA establishes parking rates for on-street parking meters. Before the project started, parking rates were the same all day, every day, independent of the parking demand. By implementing a demand responsive pricing scheme, parking availability improved dramatically. The facts listed in Table 3.1 show the improvements on parking after the project completed.

▷ the amount of time parking areas were too full decreased by 16%
▷ the amount of time to find a parking space decreased by 43%
▷ parking citations decreased by 23%
▷ traffic volume decreased by 8% in the pilot areas
▷ the average hourly rate was overall decreased by 11 cents, from \$2.69 to \$2.58
▷ double-parking decreased by 22%
▷ there was a 30% drop of greenhouse gas emissions per day (originally 7 cubic meters tons)
▷ there was an increase in \$1.9M annual revenue, although raising revenue was not a goal of the project

Table 3.1.: The main results from the SF*park* project

In conducting the project, 9 pilot areas were chosen for monitoring. Out of these areas, 7 were selected to have new pricing policies, while 2 were control areas. The number of metered spaces used was 6000, which amounts for 25% of the city's total.

The meters allow rates to be deployed remotely and they transmit data to a central server through a wireless connection.

The data was collected using parking sensors. These provided the central server with the information needed to calculate the demand-responsive parking rates and provided real-time parking availability information. A parking sensor is a magnetometer that detects changes in the earth's electromagnetic field. A total of 11700 sensors were deployed, resulting in 8000 spaces that were equipped with one or two sensors. The sensors delivered valid data from April, 2011 to December, 2013. The issues with sensors were noise from the environment, which reflected in the electromagnetic interference, early battery degradation, and street construction.

SF*park* made available real-time information on parking rates and parking occupancy though a smartphone app. See Figure 3.9 for a view from the SF*park* website.

As the main provider of data used in this work, the SF*park* project and its success played an important role when choosing to base our project on it.

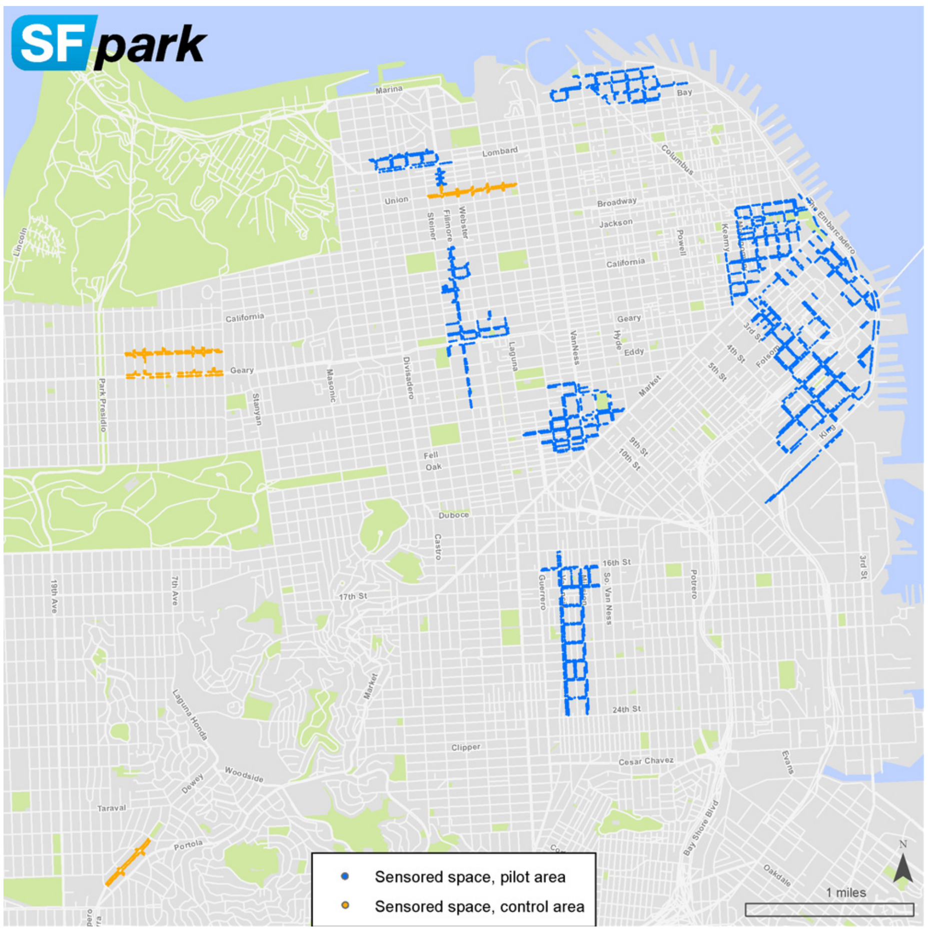


Figure 3.9.: An overview of the SF*park* map [31]

4. Related Work

Smart parking and its connected field of smart city are growing fields that currently produce a significant number of academic publications. In this chapter we shall briefly describe the research in the area of smart parking, while concentrating on the status quo of parking prediction systems.

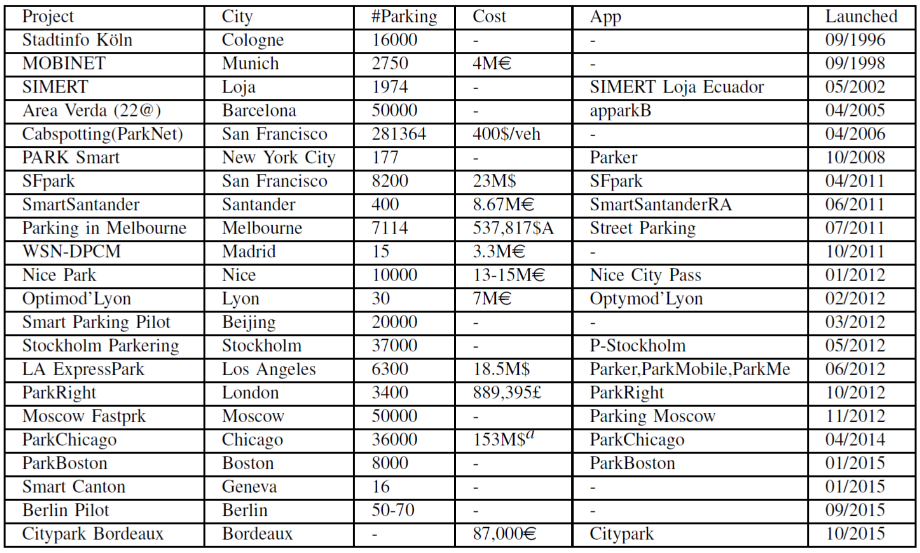
4.1. Smart Parking Overview

With the rapidly emerging technology, especially after year 2000, the field of smart parking has risen within the area of smart city. Smartphones and vehicle-integrated navigation systems have made access to city street information very convenient to the user. On the other end, the smart infrastructure is able to deliver alerts whenever a road was blocked or a traffic event is happening.

A significant part of what makes a city smart is parking. Being able to have an overview of parking spaces availability, either on-street or in a car park, makes driving in the city much easier. A person may decide from home, when looking at such an overview, whether to take the car in the city or not, based on the chances that they have to find a parking space. For an overview of smart parking projects, including development cost and the number of parking spaces covered, see Figure 4.1.

In the scientific literature there have been an increased number of publications on smart parking. We only touch upon some topics, such as parking reservation and resource allocation, our main focusing being occupancy prediction. We refer to a recent smart parking survey completed in 2017, that presents the developments in the field. The survey [57] is a follow-up to the main author's PhD thesis published in 2015 on the same topic [58]. It presents the relevant developments on smart parking solutions since 2000. We shall use this survey to briefly describe the research landscape. Afterwards, we will focus on parking prediction and we will go in depth on the relevant work done in that department.

As the research and development have been going in all directions, Lin et al. split the available results in *information collection*, *system deployment*, and *service dissemination*.



Project	City	#Parking	Cost	App	Launched
Stadtinfo Köln	Cologne	16000	-	-	09/1996
MOBINET	Munich	2750	4M€	-	09/1998
SIMERT	Loja	1974	-	SIMERT Loja Ecuador	05/2002
Area Verda (22@)	Barcelona	50000	-	apparkB	04/2005
Cabspotting(ParkNet)	San Francisco	281364	400$/veh	-	04/2006
PARK Smart	New York City	177	-	Parker	10/2008
SFpark	San Francisco	8200	23M$	SFpark	04/2011
SmartSantander	Santander	400	8.67M€	SmartSantanderRA	06/2011
Parking in Melbourne	Melbourne	7114	537,817$A	Street Parking	07/2011
WSN-DPCM	Madrid	15	3.3M€	-	10/2011
Nice Park	Nice	10000	13-15M€	Nice City Pass	01/2012
Optimod'Lyon	Lyon	30	7M€	Optymod'Lyon	02/2012
Smart Parking Pilot	Beijing	20000	-	-	03/2012
Stockholm Parkering	Stockholm	37000	-	P-Stockholm	05/2012
LA ExpressPark	Los Angeles	6300	18.5M$	Parker,ParkMobile,ParkMe	06/2012
ParkRight	London	3400	889,395£	ParkRight	10/2012
Moscow Fastprk	Moscow	50000	-	Parking Moscow	11/2012
ParkChicago	Chicago	36000	153M$[a]	ParkChicago	04/2014
ParkBoston	Boston	8000	-	ParkBoston	01/2015
Smart Canton	Geneva	16	-	-	01/2015
Berlin Pilot	Berlin	50-70	-	-	09/2015
Citypark Bordeaux	Bordeaux	-	87,000€	Citypark	10/2015

Figure 4.1.: Large-scale smart parking projects [57]

Under information collection, the authors include all techniques to acquire parking information. Be it static or mobile, in the form of sensors or as parking meters, the infrastructure here is diverse. Most of the time, the deployed sensors are managed through a wireless network infrastructure. The sensors are present inside cars, such as taxis, which travel through the city and collect roadside information on parking. The piece of information transmitted is an occupancy change bit: either the car has left or arrived at a parking space. A number of types of sensors are usually being used in the process: infrared sensors, ultrasonic sensors, accelerators, optical sensors, inductive loops, piezoelectric sensors, cameras, acoustic sensors. Most of the time, the captured data requires post-processing in the form of image or audio recognition before arriving at the target occupancy information. Some of the captured information also raises privacy issues, at it contains sensitive data about the car and driver. Smartphones provide the means to collect data and can have a great impact through crowdsourcing, if the users are given incentives to enable the respective smartphone functions. On the other hand, there are systems where the driver initiates the data transmission. By using a mobile application, the driver may choose to report that he or she just left a parking space, respectively has arrived at a parking space.

In the system deployment section, the survey refers to the varieties of parking systems, looks into how well they scale, and touches upon their data analysis side. The parking systems software is the interface between the data sources and the users. Software systems are often in the form reservation systems, typically run by municipalities. Other systems also may provide guiding assistance in arriving at the desired parking space. Their vacancy prediction component informs the drivers about the availability of parking spaces at the destination. In this way, they contribute to the decision of the driver, of whether to take the car to the destination or not. Prediction systems recommend parking locations that have at least one parking space free.

Under data dissemination, the authors of the survey address the ways in which parking data is exchanged between drivers. This scenario often occurs in decentralized parking systems, when the drivers find out about free parking spaces in an area where other cars drove by. Dynamic pricing is often used to control parking occupancy: parking prices are raised in areas that are almost fully occupied, whereas areas with low parking rates get assigned a lower price. A more advanced version adjusts the prices when enough demands received by the parking system would point to a future parking overload in the respective area. These approaches take into account parking competition: multiple drivers are looking to park in the same parking location.

4.2. Vacancy Prediction Systems

On the topic of prediction, there are a number of papers that approach the problem. The circumstances differ in each case, so that it is worth mentioning the papers individually first. Afterwards we will attempt to compare these approaches by extracting the most relevant aspects, e.g. method, type of data, features, error, deployment location, etc., in a table.

Xu et al. [71] makes real-time parking availability estimations based on a system that aggregates the data coming from mobile phones. The system uses algorithms based on statistical weighted schemes and Kalman filters. Additionally, the authors create parking availability profiles based on historical data and using statistical algorithms.

Chen et al. [46] develop an Android application that finds a parking location at park-and-ride facilities by calculating the probability of parking availability and taking in consideration the shortest travel time. The authors employ fuzzy logic to model the uncertainty of parking availability. The fuzzy membership function was chosen to be linear. The authors proposed multiple criteria in finding the best parking location, such as train frequency, service quality, parking-and-ride price, which should be considered in further investigations. The use case and parking data were set in Perth, Australia.

Vlahogianni et al. [68] define prediction measures to find out the duration of free parking spaces. Additionally, they calculate 1-hour parking availability forecasts. The authors use neural networks for time-series in the form of a multilayer perceptrons to accurately predict occupancy up to one hour ahead. Their findings show a Weibull distribution for the duration of free parking spaces. The system developed was incorporated into the routing service of the city of Santander, Spain.

Rajabioun and Ioannou [62] introduce an information system for parking guidance that enables communication between vehicles and the infrastructure. It proposes a prediction algorithm that forecasts the availability for parking locations based on

Paper	Year	System	Data Sources	On/Off -Street	Location
Caliskan [43]	2007	VANET	vehicle sensors	off-street	Brunswick, Germany
Klappenecker [54]	2010	VANET	vehicle sensors	off-street	-
Wu [70]	2014	parking guidance	municipality	off-street	Taipei City
Tiedemann [67]	2015	parking prediction	sensors	on-street	Berlin
Rajabioun [62]	2013	parking guidance	on-site sensors: infrared, image-processing cameras, inductor loops	both	Los Angeles, San Francisco
Rajabioun [63]	2015	parking guidance	sensors (cf. SF*park* [31])	both	Los Angeles, San Francisco
Vlahogianni [68]	2014	routing system	IoT nodes: car presence	on-street	Santander
Z. Chen [46]	2013	Location-based service for parking finding	manual collection	parking lots	Perth, Australia
Xu [71]	2013	parking prediction	mobile phone	on-street	San Francisco
X. Chen [45]	2014	parking prediction	SF*park* sensors	on-street	San Francisco
Hössinger [52]	2014	parking prediction	parking tickets manual counts traffic flow	on-street	Vienna
Caicedo [42]	2012	parking reservation	in-cars sensors	both	Barcelona
Nandugudi [60]	2014	parking prediction	smarphone	surface lots	Buffalo
Szczurek [66]	2010	VANET	GPS in car	both	-
Ji [53]	2014	parking guidance	-	off-street	Newcastle
Koster [55]	2014	parking recommender	smartphone	both	New York
Kotb [56]	2016	parking reservation	occupancy sensors	-	-
Pullola [61]	2007	parking prediction	GPS in car	on-street	Ottawa
Richter [64]	2014	-	SF*park* sensors [31]	on-street	San Francisco
Shin [65]	2014	-	sensors parking facilities cruising cars	both	Luxembourg
Zheng [72]	2015	-	parking sensors	on-street	San Francisco Melbourne

Table 4.1.: Overview of all the systems introduced in individual articles

real-time parking information. It takes into account parameters such as parking duration, arrival time, destination, pricing, walking distance, parking capacity, rates of vehicles occupying and leaving parking spots, time restrictions, parking rules, events that disrupt parking availability, etc. Their algorithm uses a probabilistic density distribution model. The parking data was collected both from on-street parking meters and off-street garages in Los Angeles and San Francisco, USA. In a following paper, Rajabioun and Ioannou [63] propose a multivariate autoregressive model that considers the temporal and spatial correlations of parking availability when making predictions (cf. Figure 4.2). The authors hold that the model, which is integrated

Paper	Approach	Features	Methods	Evaluation
Caliskan [43]	parking prediction	age of parking info, arrival time, total capacity, occupancy, arrival rate, parking rate	Markov chains	probability density, relative deviation
Klappenecker [54]	parking prediction	total capacity, occupancy, arrival rate, parking rate	Markov chains	-
Wu [70]	parking recommendation	occupancy	probabilistic	historical mean, moving average
Tiedemann [67]	parking prediction	occupancy	data threads	-
Rajabioun [62]	parking prediction	pricing, events, location, total capacity, occupancy rate, leaving rate	probabilistic	mean error
Rajabioun [63]	parking prediction	occupancy, total capacity	multivariate-autoregressive model	orthogonality error differential covariance mean absolute -percentage error
Vlahogianni [68]	parking prediction	free space duration, occupancy, traffic volume, type of day, time period	genetically optimzed, multilayer perceptron	Weibull distribution
Z. Chen [46]	parking prediction	parking times	fuzzy logic	occupancy error shortest travel time
Xu [71]	real-time estimation	parking times	Kalman adaptive linear memory filter	RMSE
X. Chen [45]	parking prediction	events, distance, price	ARIMA, linear regression, SVM, feed forward neural network	MAPE
Hössinger [52]	parking prediction	occupancy rate	linear regression, average	RMSE
Caicedo [42]	parking prediction	capacity, price, occupancy	probabilistic models	average error
Nandugudi [60]	parking prediction	parking times	probabilistic model	absolute error
Szczurek [66]	parking prediction	parking times	MALENA algorithm, Naive Bayes	average discovery time improvement
Ji [53]	parking prediction	parking times	wavelet neural network	MSE
Koster [55]	parking recommendation	parking times	Markov chains	Pearson coefficient
Kotb [56]	-	-	Mixed-integer Linear Programming	-
Pullola [61]	parking prediction	parking times	Poisson process	absolute error
Richter [64]	parking prediction	see SF*park* [33]	time series model	absolute error
Shin [65]	parking prediction	driving duration, distance, walking distance, parking cost, traffic	heuristic	average driving duration, average walking distance, parking fail rate, parking utilization rate, average occupancy
Zheng [72]	parking prediction	parking occupancy	regression trees, neural network, SVM	MSE, R^2

Table 4.2.: Overview of all the prediction methods used in related work

into a parking guidance and information system, recommends parking locations with high accuracy.

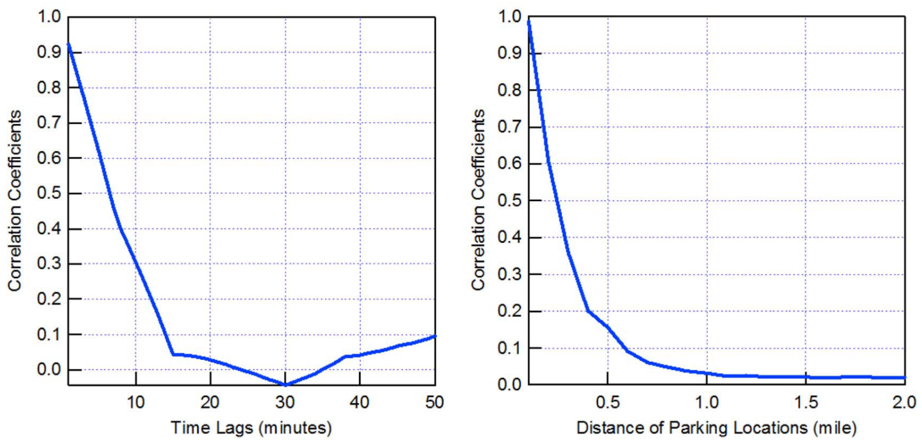


Figure 4.2.: Left: Temporal correlation for parking occupancy Right: Spatial correlation for parking occupancy. Data collected from on-street locations in San Francisco [63]

Tiedemann et al. [67] present the development of a prediction system that gives estimated occupancies for parking spaces. The occupancy data is collected online via roadside parking sensors and the prediction is realized using neural gas machine learning combined with data threads. The authors notice that some factors play a significant part in the predictions, such as holidays, weather and use the neural gas clustering to separate the data, before the data thread method is applied (cf. Figure 4.3). The system is developed by the DFKI Robotics Innovation Center, while the sensors are manufactured by Siemens AG and are installed in street lights. The pilot region for the project is Berlin, Germany.

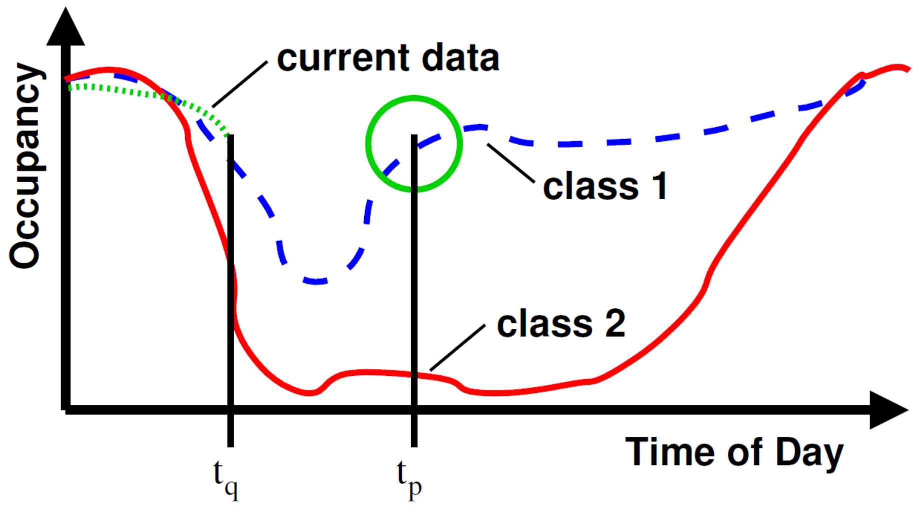


Figure 4.3.: Visualization of data threads that are used for predicting occupancy. Class 1 (dashed blue curve) and Class 2 (solid red curve) are two learned time series. When the user queries for the occupancy prediction at t_q for time t_p, the curve that is closest to the current data curve (dotted green) up to that point is chosen [67]

Wu et al. [70] present a information system that recommends parking options in the city, in order to reduce the vehicle emissions and decrease traffic. The system constructs an optimal parking sequence based on the forecasted parking availability. For the latter, probabilistic algorithms are used, i.e., shift prediction, thread estimation and probability estimation. The data and experimental results have been carried out in Taipei City in 2011.

Caliskan et al. [43] model the prediction of available parking spaces as a vehicular ad-hoc network (VANET). The network disseminates parking data in order to help with the estimation of future occupancy of parking lots. The pieces of disseminated data are timestamp, total capacity of parking lot, number of parking spaces that are currently occupied, the arrival rate, and the parking rate. The latter two are used in the modeling of continuous-time homogeneous Markov chains. The approach is otherwise based on queuing theory. The evaluation was carried out in Brunswick, Germany, where about 10000 vehicles were connected in the ad-hoc network. Klappenecker et al. [54] builds on the result of Caliskan and uses an improved version of continuous-time Markov chains for predicting availability of parking spaces. Predictions are communicated between cars in an ad-hoc network. The approach simplifies the computations of transitional probabilities inside a Markov chain model. The system applies to parking lots that are connected to the ad-hoc network. These communicate the number of occupied spaces, capacity, arrival and parking rate. Also based on VANETs is Szczurek et al. [66] work, which propose a novel approach that combines machine learning with the information disseminated in ad-hoc vehicular networks. The building block of the system are parking reports, which are issued by vehicles leaving a parking space and comprise a report identifier, a location, and a timestamp. The parking reports are being learned by a model, which then indicates whether a parking is available for a specific vehicle. A conditional relevance is used to determine whether a particular report is useful for a specific vehicle. This is modeled using a Naive Bayes method. A parking availability report R is labeled relevant by vehicle V, if the parking space referenced in R is available when V reaches it. Upon evaluation of the methods, the authors reported an improvement in parking discovery times for vehicles.

PocketParker is a crowdsourcing system, proposed by Nandugudi et al. [60], that uses smartphone data to predict parking availability. The system is used for parking lots. It requires no input from the user, it notices automatically when a user starts to drive or stops, i.e., departure and arrival events (cf. Figure 4.4). Based on these two events, the system builds a probability distribution model that is used to answer queries about parking availability. PocketParker has proved robust to hidden parkers, i.e., parking vehicles that are not using the application. In the authors' simulation, it has reached 94% rate for parking availability prediction with 105 users over 45 days.

Caicedo et al. [42] develop a methodology for predicting real-time parking space availability. The probabilistic algorithm consists of three subroutines: allocating simulated parking requests, estimating future departures, and forecasting parking availability. The forecast has been reported to improve as the system registers arrivals

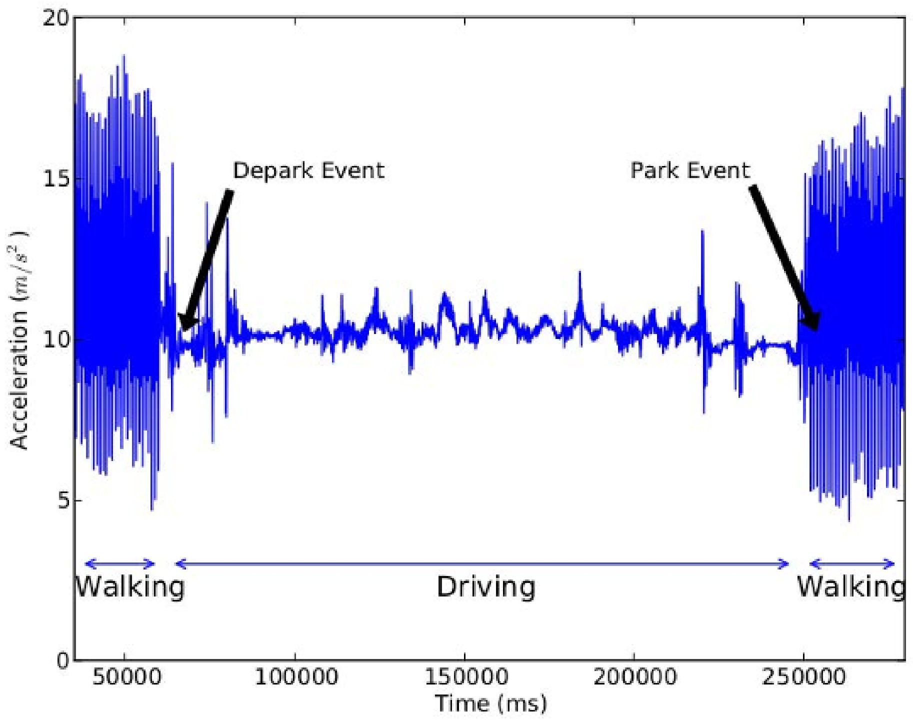
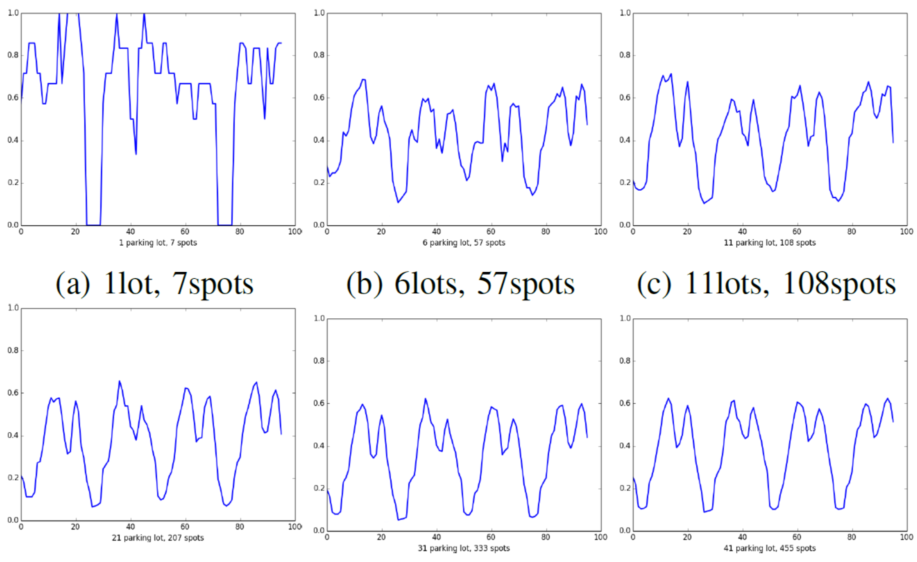


Figure 4.4.: PocketParker detects a parking event using accelerometer data from the driver's smartphone [60]

and departures. Further factors taken into account were duration of stay and capacity of every operating parking facility. The system was tested in Barcelona, Spain, with very satisfactory results.

Hössinger et al. [52] present a simple real-time occupancy model based on various pieces of data collected in the city of Vienna, Austria. An average day curve model was built using the ticket data from mobile-phone parking, the counts of car parks, and the traffic flow volumes in the city. The data was collected following agreements with the respective mobile phone companies, through surveys and by accessing a dedicated traffic website, respectively. The predictions are valid for short-time spans and applicable to the above mentioned city.

Chen [45] tackles the parking problem by aggregating parking lots. The paper shows that the prediction error of parking occupancy decreases by combining multiple parking lots. The trained models take into account factors such as day, time, event, distance, parking price, etc. The author tries out multiple models, such as ARIMA, linear regression, support vector regression and feed forward neural network. It turns out that the neural networks algorithm scores the best when the model is evaluated on the SF*park* data [32]. Furthermore, the paper investigates the specifics of aggregated parking lots (cf. Figure 4.5) and finds certain artifacts in the occupancy graph when clustering the respective parking locations.



(a) 1lot, 7spots (b) 6lots, 57spots (c) 11lots, 108spots

(d) 21 lots, 207spots (e) 31lots, 333spots (f) 41lots, 455spots

Figure 4.5.: Increasing aggregation levels of parking lots. The more parking spaces are included, the smoother is the emerging pattern [45]

Richter et al. [64] address the parking prediction problem with the focus on model storage in vehicles. The authors train models of various granularity that would predict parking availability based on the information contained: a one-day model per road segment, a three-day model per road segment, and a seven-day model per road segment. Additionally, models based on regions and time intervals computed by clustering are tried out. Hierarchical clustering with complete linkage is employed. The models are evaluated on street data from the SF*park* project [32]. The application of clustering before building the models shows a 99% decrease of model storage space. The prediction success rate is at about 70%.

With iParker, Kotb et al. [56] propose a system that handles parking reservations. It achieves resource allocation so that drivers will pay less for parking, while parking managers receive more resource utilization and hence reach higher revenue. The system is based on mixed-linear programming (MILP). The system uses dynamic resource allocation and pricing models to achieve its goal. At its evaluation, it was reported to cut the cost for drivers by 28%, achieving a 21% increase in resource utilization, and it increased the total revenue for parking management by 16%. Although this work only indirectly deals with parking occupancy, it makes use of it in a system that could include prediction models in the future.

ParkNet, developed by Mathur et al. [59] is a system made up of vehicles that captures parking space information while driving. Every ParkNet vehicle is equipped with a GPS receiver and an ultrasonic sensor facing sideways (cf. Figure 4.6). The latter determines whether it passes by parking spaces and whether they are occupied. The data is sent to a central server that aggregates it, in order to build parking space

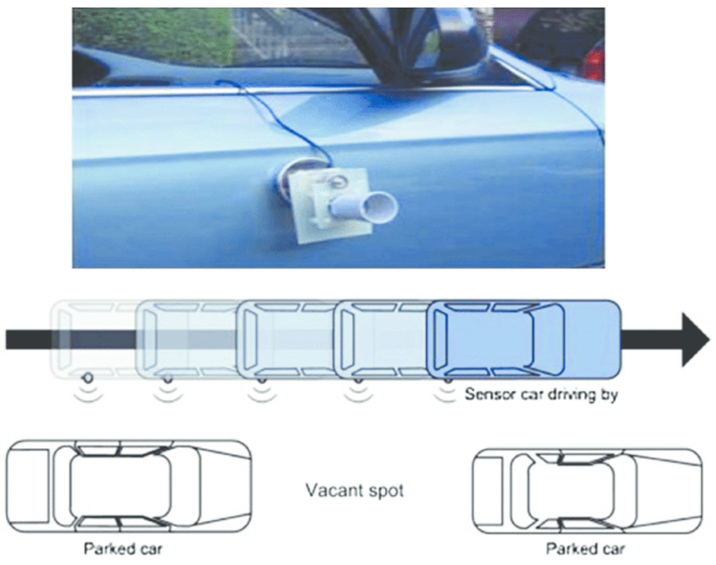


Figure 4.6.: The ultrasonic sensor mounted sideways on cars in the ParkNet system [59]

occupancy maps in real-time. The information is queried by clients that search for a free parking space. The system was evaluated in Highland Park, New Jersey and San Francisco on 500 miles road-side parking data and yield 95% accurate parking maps and 90% parking occupancy accuracy. The authors show that the system can further be improved if the sensors are fitted into taxicabs or city buses.

Ji et al. [53] presents a forecasting method based on wavelet neural networks. These are feed-forward neural networks that have continuous wavelet function as activation function for the hidden layers. The system was tested in Newcastle upon Tyne, England and evaluation was performed using real-time data of up to 1 day in advance.

Koster et al. [55] propose a smartphone-based solution that recognizes when drivers arrive or leave parking spaces. A “Bayesian approach” and Hidden Markov Models (HMM) are used to model the parking spaces and respond to user queries for the next parking space. The HMM are based on gathered historical data. As answer to the user query is a parking space nearby and the probability of it being free at the respective time. The authors emphasize the non-intrusive nature of the solution, where drivers only have to minimally interact with their phones to get a recommended free parking space.

Pullola et al. [61] propose a solution to determine the availability of parking lots by modeling the occupancy as a non-homogeneous Poisson process. Past occupancy information of the parking lots is stored and leveraged. The data and the computation are made inside the GPS system of the vehicle, in order not to depend on the quality of the transmitted signal.

Zheng et al. [72] investigates the results of a proposed prediction mechanism on the parking data collected in San Francisco and Melbourne. The three algorithms proposed employ regression trees, support vector regression and neural networks. The feature sets included the historical occupancies alongside the time and day of the week.

Shin and Jun [65] propose an algorithm for smart parking that assigns cars to parking facilities in the city. The criteria based on which the assigning is realized includes driving distance to the parking facility, walking distance from the parking facility to the destination, parking cost, and traffic congestion. The real-time data is collected from parking facilities and from sensors that are integrated in cruising cars. The data is transferred from the central server, where it is managed through a wired/wireless telecommunication network. The authors test their approach in Luxembourg City. The results of the simulations showed improved figures for average driving duration, average walking distance, parking failing rate, parking utilization rate, average standard deviation on the number of guided cars to each parking facility, average occupancy ratio of parking facility, and for the parking facility occupancy rate.

In order to summarize the relevant related literature, we have gathered the benchmark data into two tables. Table 4.1 shows an overview of the systems presented in the papers, alongside the data sources used, whether the system is dedicated to on- or off-street parking, and the location where it was evaluated. Table 4.2, on the other hand, emphasizes the prediction character of the papers by highlighting their general approach, features, methods, and evaluation.

5. Design & Implementation

This chapter describes the approach that solves the problem defined in Chapter 1. The solution detailed here is a standard one, agnostic to any particular city and use case, as our goal is to have it suitable for various scenarios. Both theoretical and implementation aspects will be discussed.

We start by defining the parking data used, followed by the city data. Both types of data are processed in RDF format. The merging of the two sets of data is shortly discussed.

Furthermore, the clustering process is tackled in the smart parking setting. The parking profile is concretely defined as an urban measure, to which two representations are assigned: cosine vectors and EMD Gaussians. These lead us to defining the similarity functions: cosine similarity and EMD. Finally, the machine learning training process is explained and the estimations for clusters without parking data are defined.

A brief account of implementation aspects is also given, including database organization and data processing. An overview of the process can be visualized in Figure 5.1.

5.1. Parking Data

In solving the parking problem, the parking data is essential. In our case, as outlined in Section 2.2, we need parking data for estimating the parking occupancy in the city.

Parking data typically is found as comma-separated-value (CSV) files. We assume it to generally contain the following information.

1. *Parking occupancy* contains information on the availability of parking spaces. See Section 5.1.1.

2. *Traffic data* contains information regarding the city traffic, which is relevant for parking. See Section 5.1.2.

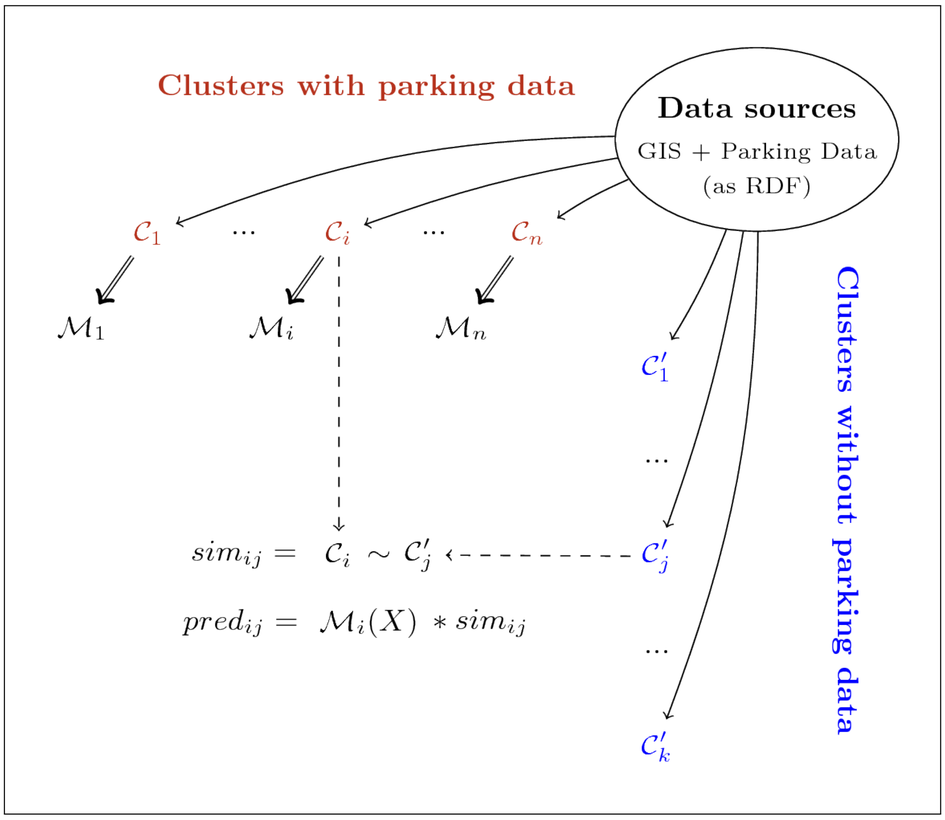

Clusters with parking data
Data sources
GIS + Parking Data
(as RDF)
$\mathcal{C}_1$ … $\mathcal{C}_i$ … $\mathcal{C}_n$
$\mathcal{M}_1$
$\mathcal{M}_i$
$\mathcal{M}_n$
$\mathcal{C}'_1$
$\mathcal{C}'_j$
$\mathcal{C}'_k$
$sim_{ij} = \mathcal{C}_i \sim \mathcal{C}'_j$
$pred_{ij} = \mathcal{M}_i(X) * sim_{ij}$
Clusters without parking data

Figure 5.1.: Concept Visualization

3. *Weather data* contains weather information for the same area as for the parking problem. See Section 5.1.4.

4. *Event data* contains event information which may have an impact on parking. See Section 5.1.3.

5. *Parking revenue data* contains economic information on parking, whose relevance may influence parking prediction. See Section 5.1.6.

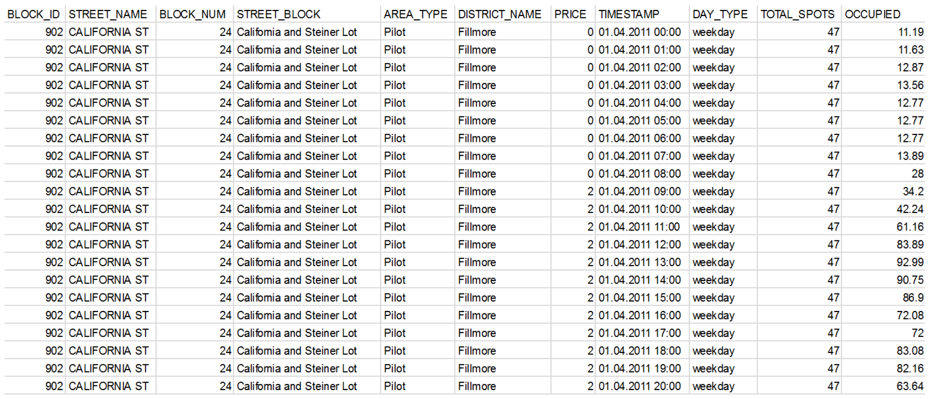
6. *Fuel price data* contains prices of fuel in the region where our problem lies. See Section 5.1.5.

Alongside the parking data above, we expect to have a infrastructure map that identifies the locations in the city, the *location data*. It contains location ids and their corresponding geometries. This may be available either as shape file or as CSV file. Tabular information can be extracted from shape files using GIS applications, such a QGIS or ArcGIS. All the pieces of information enumerated above will reference the locations found in this file.

5.1.1. Parking Occupancy Data

Parking occupancy is typically available in comma-separated-value (CSV) format. See Figure 5.2 for an example snippet. For convenience, we have included the parking price information here. A row from such a file typically contains the following pieces of information:

- ▷ location unit id – usually expressed as block id, street name, or district name. This information references the location id data. The lower the level of the location unit, the more precise the data is.
- ▷ date – expressed in a particular date format.
- ▷ time – usually as full hours. However, parking data may also be provided when a car has just parked or left. The time is sometimes delivered together with the date, as timestamp. Note that it is likely that the parking occupancy data has not been recorded over a continuous time period, hence some time intervals might be missing.
- ▷ parking capacity – the total number of parking spaces at the given location.
- ▷ parking price – the price of a ticket at the certain location and the given time. Values are expressed in US dollars, Euro or any other currency.
- ▷ parking occupancy – expressed either as rate (subunitary fraction or percent) or in absolute numbers.



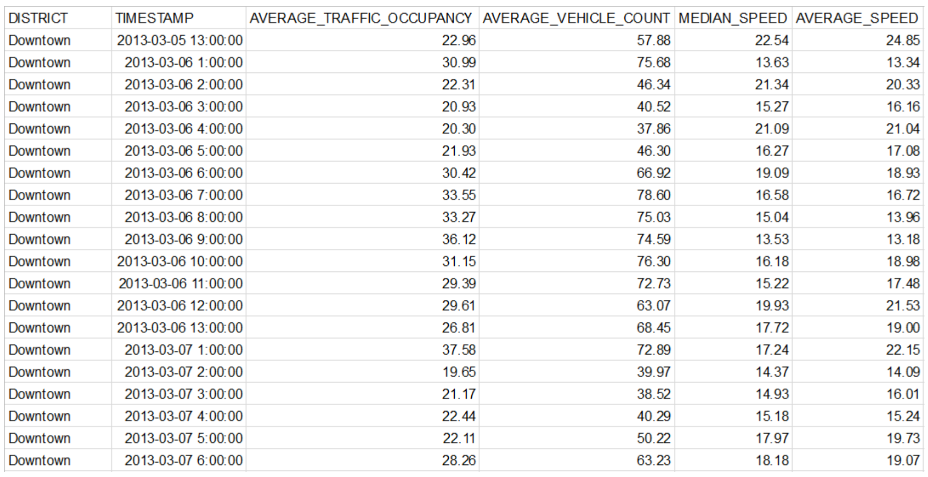
BLOCK_ID	STREET_NAME	BLOCK_NUM	STREET_BLOCK	AREA_TYPE	DISTRICT_NAME	PRICE	TIMESTAMP	DAY_TYPE	TOTAL_SPOTS	OCCUPIED
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	0	01.04.2011 00:00	weekday	47	11.19
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	0	01.04.2011 01:00	weekday	47	11.63
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	0	01.04.2011 02:00	weekday	47	12.87
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	0	01.04.2011 03:00	weekday	47	13.56
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	0	01.04.2011 04:00	weekday	47	12.77
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	0	01.04.2011 05:00	weekday	47	12.77
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	0	01.04.2011 06:00	weekday	47	12.77
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	0	01.04.2011 07:00	weekday	47	13.89
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	0	01.04.2011 08:00	weekday	47	28
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	2	01.04.2011 09:00	weekday	47	34.2
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	2	01.04.2011 10:00	weekday	47	42.24
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	2	01.04.2011 11:00	weekday	47	61.16
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	2	01.04.2011 12:00	weekday	47	83.89
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	2	01.04.2011 13:00	weekday	47	92.99
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	2	01.04.2011 14:00	weekday	47	90.75
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	2	01.04.2011 15:00	weekday	47	86.9
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	2	01.04.2011 16:00	weekday	47	72.08
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	2	01.04.2011 17:00	weekday	47	72
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	2	01.04.2011 18:00	weekday	47	83.08
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	2	01.04.2011 19:00	weekday	47	82.16
902	CALIFORNIA ST	24	California and Steiner Lot	Pilot	Fillmore	2	01.04.2011 20:00	weekday	47	63.64

Figure 5.2.: Snippet of SF*park* parking occupancy data

5.1.2. Traffic Data

The traffic data generally contains information about the volume of traffic at a certain location, at a certain time. See Figure 5.3 for an example snippet.

- ▷ location unit id – the same as for Section 5.1.1, ideally at the same granularity level.
- ▷ date and time – recorded usually at full hours or in periodic time intervals. Ideally, the times coincide at least partially with the times from the parking occupancy data.
- ▷ traffic value – typically expressed as average traffic road occupancy, average vehicle count, median speed, or average speed of the travelling cars.



DISTRICT	TIMESTAMP	AVERAGE_TRAFFIC_OCCUPANCY	AVERAGE_VEHICLE_COUNT	MEDIAN_SPEED	AVERAGE_SPEED
Downtown	2013-03-05 13:00:00	22.96	57.88	22.54	24.85
Downtown	2013-03-06 1:00:00	30.99	75.68	13.63	13.34
Downtown	2013-03-06 2:00:00	22.31	46.34	21.34	20.33
Downtown	2013-03-06 3:00:00	20.93	40.52	15.27	16.16
Downtown	2013-03-06 4:00:00	20.30	37.86	21.09	21.04
Downtown	2013-03-06 5:00:00	21.93	46.30	16.27	17.08
Downtown	2013-03-06 6:00:00	30.42	66.92	19.09	18.93
Downtown	2013-03-06 7:00:00	33.55	78.60	16.58	16.72
Downtown	2013-03-06 8:00:00	33.27	75.03	15.04	13.96
Downtown	2013-03-06 9:00:00	36.12	74.59	13.53	13.18
Downtown	2013-03-06 10:00:00	31.15	76.30	16.18	18.98
Downtown	2013-03-06 11:00:00	29.39	72.73	15.22	17.48
Downtown	2013-03-06 12:00:00	29.61	63.07	19.93	21.53
Downtown	2013-03-06 13:00:00	26.81	68.45	17.72	19.00
Downtown	2013-03-07 1:00:00	37.58	72.89	17.24	22.15
Downtown	2013-03-07 2:00:00	19.65	39.97	14.37	14.09
Downtown	2013-03-07 3:00:00	21.17	38.52	14.93	16.01
Downtown	2013-03-07 4:00:00	22.44	40.29	15.18	15.24
Downtown	2013-03-07 5:00:00	22.11	50.22	17.97	19.73
Downtown	2013-03-07 6:00:00	28.26	63.23	18.18	19.07

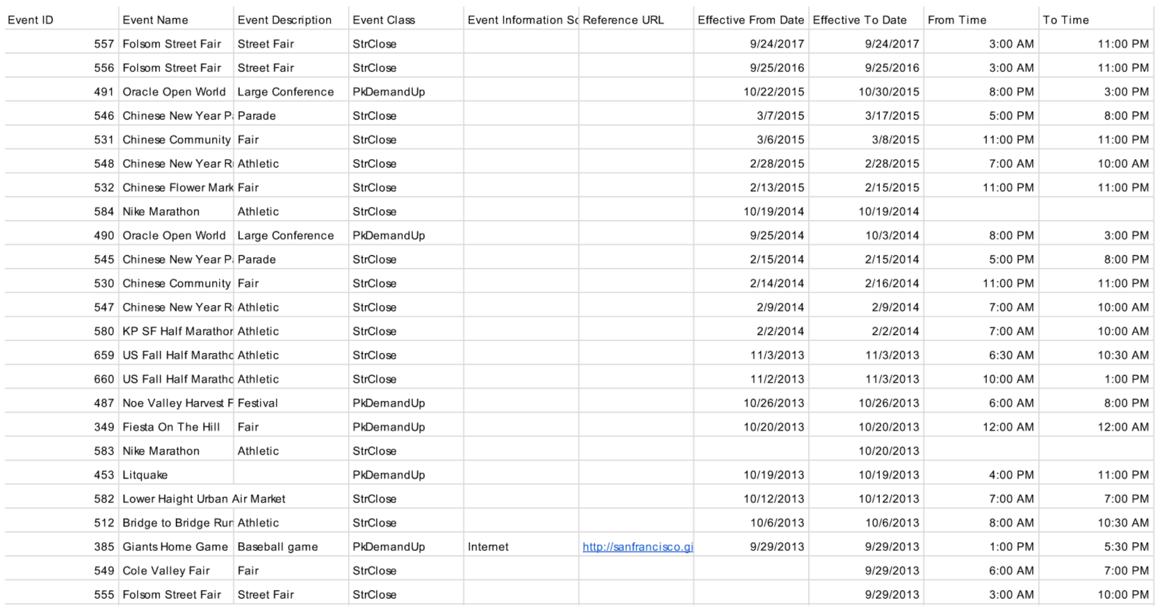
Figure 5.3.: Snippet of SF*park* traffic data

5.1.3. Events Data

Data on events is the information that the municipality makes public, when road blocks occur due to open air events. It may also include construction work, or is an indication that the parking demand is rising because of a concert, sports game, etc. See Figure 5.4 for an example snippet.

- ▷ location unit id – the same as for Section 5.1.1, ideally at the same granularity level.
- ▷ date and time interval – expressing the time interval during which a road is closed or is believed to be subject to sudden rise of parking demand.

- ▷ event name class – the name of the event and its class (road closure, rise of parking demand).



Event ID	Event Name	Event Description	Event Class	Event Information Sc	Reference URL	Effective From Date	Effective To Date	From Time	To Time
557	Folsom Street Fair	Street Fair	StrClose			9/24/2017	9/24/2017	3:00 AM	11:00 PM
556	Folsom Street Fair	Street Fair	StrClose			9/25/2016	9/25/2016	3:00 AM	11:00 PM
491	Oracle Open World	Large Conference	PkDemandUp			10/22/2015	10/30/2015	8:00 PM	3:00 PM
546	Chinese New Year P	Parade	StrClose			3/7/2015	3/17/2015	5:00 PM	8:00 PM
531	Chinese Community	Fair	StrClose			3/6/2015	3/8/2015	11:00 PM	11:00 PM
548	Chinese New Year R	Athletic	StrClose			2/28/2015	2/28/2015	7:00 AM	10:00 AM
532	Chinese Flower Mark	Fair	StrClose			2/13/2015	2/15/2015	11:00 PM	11:00 PM
584	Nike Marathon	Athletic	StrClose			10/19/2014	10/19/2014		
490	Oracle Open World	Large Conference	PkDemandUp			9/25/2014	10/3/2014	8:00 PM	3:00 PM
545	Chinese New Year P	Parade	StrClose			2/15/2014	2/15/2014	5:00 PM	8:00 PM
530	Chinese Community	Fair	StrClose			2/14/2014	2/16/2014	11:00 PM	11:00 PM
547	Chinese New Year R	Athletic	StrClose			2/9/2014	2/9/2014	7:00 AM	10:00 AM
580	KP SF Half Marathor	Athletic	StrClose			2/2/2014	2/2/2014	7:00 AM	10:00 AM
659	US Fall Half Maratho	Athletic	StrClose			11/3/2013	11/3/2013	6:30 AM	10:30 AM
660	US Fall Half Maratho	Athletic	StrClose			11/2/2013	11/3/2013	10:00 AM	1:00 PM
487	Noe Valley Harvest F	Festival	PkDemandUp			10/26/2013	10/26/2013	6:00 AM	8:00 PM
349	Fiesta On The Hill	Fair	PkDemandUp			10/20/2013	10/20/2013	12:00 AM	12:00 AM
583	Nike Marathon	Athletic	StrClose				10/20/2013		
453	Litquake		PkDemandUp			10/19/2013	10/19/2013	4:00 PM	11:00 PM
582	Lower Haight Urban Air Market		StrClose			10/12/2013	10/12/2013	7:00 AM	7:00 PM
512	Bridge to Bridge Run	Athletic	StrClose			10/6/2013	10/6/2013	8:00 AM	10:30 AM
385	Giants Home Game	Baseball game	PkDemandUp	Internet	http://sanfrancisco.gi	9/29/2013	9/29/2013	1:00 PM	5:30 PM
549	Cole Valley Fair	Fair	StrClose				9/29/2013	6:00 AM	7:00 PM
555	Folsom Street Fair	Street Fair	StrClose				9/29/2013	3:00 AM	10:00 PM

Figure 5.4.: Snippet of SF*park* events data

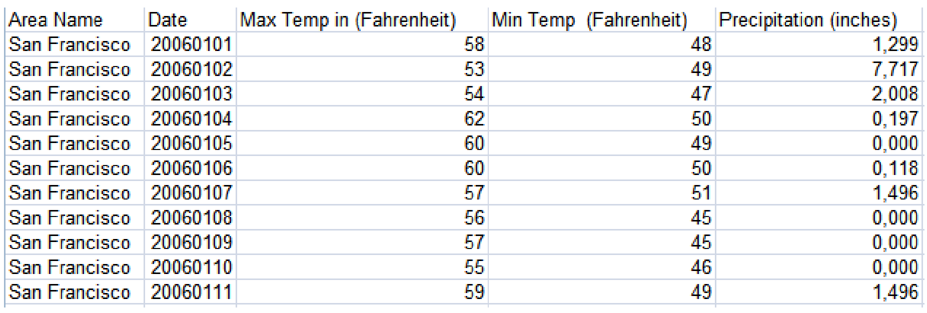
5.1.4. Weather Data

The weather data generally contains values for temperatures and precipitation for a certain city on a certain day. See Figure 5.5 for an example snippet.

- ▷ location unit id – the scope here is usually a whole city or a suburb.
- ▷ date and time – either hourly or for 24 hours intervals.
- ▷ temperature – depending on the time intervals, it may be current temperatures. If the scope is a whole day, it is usually the maximum and minimum temperatures that are provided. The values are expressed in either Celsius or Fahrenheit.
- ▷ precipitation – expressing the quantity of rain or snow for the corresponding time interval. The value is provided in millimeters or inches.

5.1.5. Fuel Price Data

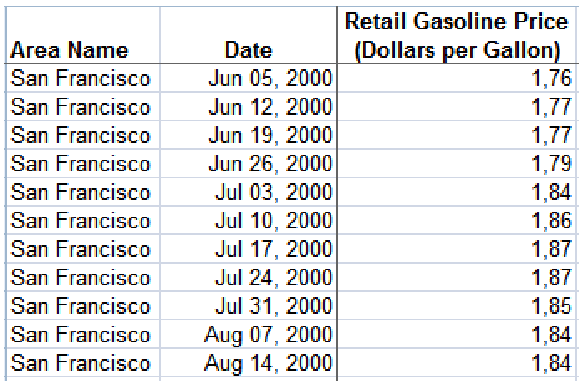
The fuel price contains the prices of gasoline or diesel in a city region on a certain day. See Figure 5.6 for an example snippet.



Area Name	Date	Max Temp in (Fahrenheit)	Min Temp (Fahrenheit)	Precipitation (inches)
San Francisco	20060101	58	48	1,299
San Francisco	20060102	53	49	7,717
San Francisco	20060103	54	47	2,008
San Francisco	20060104	62	50	0,197
San Francisco	20060105	60	49	0,000
San Francisco	20060106	60	50	0,118
San Francisco	20060107	57	51	1,496
San Francisco	20060108	56	45	0,000
San Francisco	20060109	57	45	0,000
San Francisco	20060110	55	46	0,000
San Francisco	20060111	59	49	1,496

Figure 5.5.: Snippet of SF*park* weather data

- ▷ location unit id – usually given as city.
- ▷ date and time – usually provided as day.
- ▷ type of fuel – gasoline, diesel, etc.
- ▷ price per unit – provided as the price per liter or per gallon.



Area Name	Date	Retail Gasoline Price (Dollars per Gallon)
San Francisco	Jun 05, 2000	1,76
San Francisco	Jun 12, 2000	1,77
San Francisco	Jun 19, 2000	1,77
San Francisco	Jun 26, 2000	1,79
San Francisco	Jul 03, 2000	1,84
San Francisco	Jul 10, 2000	1,86
San Francisco	Jul 17, 2000	1,87
San Francisco	Jul 24, 2000	1,87
San Francisco	Jul 31, 2000	1,85
San Francisco	Aug 07, 2000	1,84
San Francisco	Aug 14, 2000	1,84

Figure 5.6.: Snippet of SF*park* fuel price data

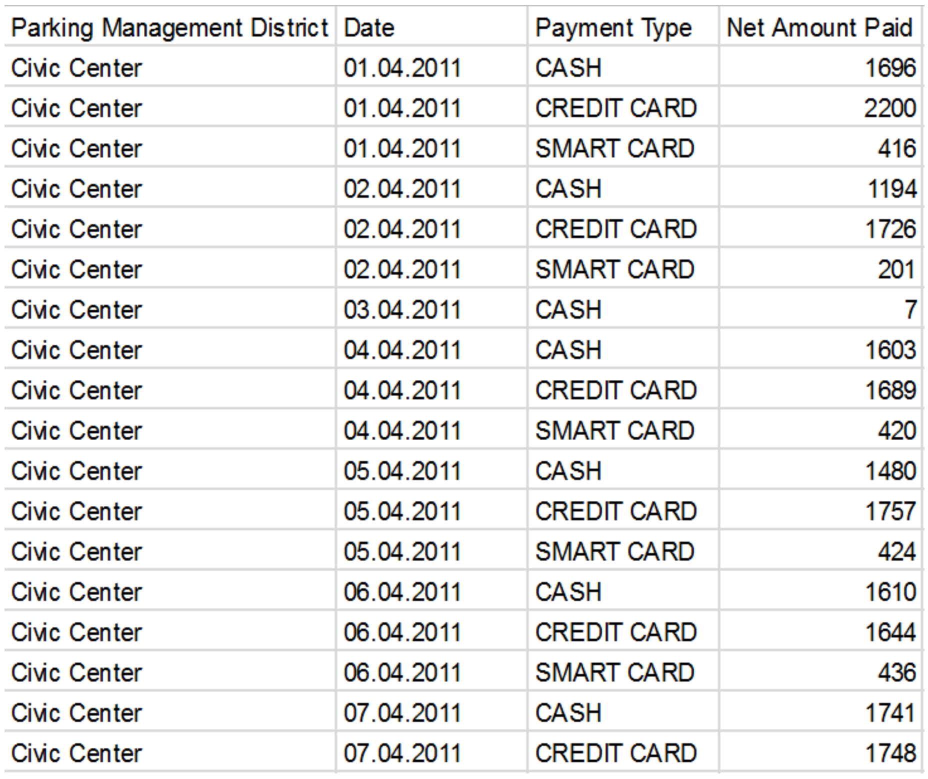
5.1.6. Parking Revenue Data

Parking revenue data provides information on the amount of money being raised from parking in a certain area. See Figure 5.7 for an example snippet. The data is usually comprised of:

- ▷ location unit id – either the area belonging to a parking meter, a whole neighborhood, or a larger parking area.
- ▷ date and time – typically provided per day.

▷ payment type – the way the driver opted to pay for parking, e.g., cash, credit card, etc.

▷ payed amount – the amount in US dollars, Euro or other currency.



Parking Management District	Date	Payment Type	Net Amount Paid
Civic Center	01.04.2011	CASH	1696
Civic Center	01.04.2011	CREDIT CARD	2200
Civic Center	01.04.2011	SMART CARD	416
Civic Center	02.04.2011	CASH	1194
Civic Center	02.04.2011	CREDIT CARD	1726
Civic Center	02.04.2011	SMART CARD	201
Civic Center	03.04.2011	CASH	7
Civic Center	04.04.2011	CASH	1603
Civic Center	04.04.2011	CREDIT CARD	1689
Civic Center	04.04.2011	SMART CARD	420
Civic Center	05.04.2011	CASH	1480
Civic Center	05.04.2011	CREDIT CARD	1757
Civic Center	05.04.2011	SMART CARD	424
Civic Center	06.04.2011	CASH	1610
Civic Center	06.04.2011	CREDIT CARD	1644
Civic Center	06.04.2011	SMART CARD	436
Civic Center	07.04.2011	CASH	1741
Civic Center	07.04.2011	CREDIT CARD	1748

Figure 5.7.: Snippet of SF*park* parking revenue data

5.2. City Data

In Section 2.1 we established that city data reflects parking demand in a city area. The plan is to collect public amenity information, as provided in OpenSteetMap (cf. Section 3.8), and enrich it with visit duration values, available from Google Places (cf. Section 3.9).

The OpenStreetMap data is available as layers. The layers are stored together in an .osm file, which is typically downloaded from OSM. Extracting the single layers involves using a tool called *osm2pgsql* [26], which imports the POIs in a database table. For our purpose, the points layer will suffice, as the POIs contain the public amenity information we are after.

5.3. RDF Annotation

In Chapter 1 we emphasized that our approach should be relevant for any city that wants to solve its parking problem. We will consider all input data used in the present approach to be in RDF format, in order to establish a common format that can be used for other cases.

When the actual parking- and city data is only available as raw values, we need to annotate it as RDF in the first place. The default process involves the extraction of data, which is afterwards available for further processing. We shall use Apache Jena to encode the parking and city data. The ontologies used in the process belong to the CityPulse project [2]. The extraction of the data will be performed using SPARQL queries.

5.4. Merging City and Park Data

In order to reference the parking- and city data from and to each other, both sets of data should have a common location unit. For parking data, the the location units are provided in the location dataset. The city data references the POI geometries, which are points expressed in a particular reference system. We first need to establish the coordinate systems of both geometries and afterwards define a *merge distance* that would match a parking space to a public amenity. In fact, the relationship should be one to many. As this process depends on the concrete data, we shall continue it later on.

5.5. Clustering

The available parking data does not cover the whole city surface. In fact, it is a fraction of it and, as assumed in Section 1.2. We shall conceptually separate the area *with* parking data from the area *without* parking data. The reason for clustering is, as argued in Section 2.2, that splitting both these regions into smaller areas leads to more representative parking profiles and parking estimations later on. As we want an exclusively location-based separation, we shall employ K-Means to cluster the city regions.

There are two clustering processes, one for the city area *with* parking data, another one for the city area *without* parking data. The number of the clusters chosen in each area should be kept proportional to the number of total location units that each city area contains.

Implementation-wise, a Python module will perform K-Means++ by using the *scikit-learn* package.

5.6. Urban Measure

An expression of the parking demand of a city area can be obtained using city data, as already established in Section 2.1. We shall use both the public amenity information from OpenStreetMap and the visit duration values from Google Places to define *urban measures* for each cluster. Urban measures have two representations: cosine vectors and EMD Gaussians.

5.6.1. Cosine Vectors

To form the cosine vectors, we shall first divide all amenities into categories $Cat_1, Cat_2, ..., Cat_n$. The criteria for division will be their average visit duration. For example, a short duration category of up to 30 minutes, a medium duration between 31 and 90 minutes and a large duration of above 90 minutes stay. Each cluster gets represented by an n-dimensional vector, whose components correspond to the amenity categories. The magnitude of component i is equal to the number of amenities of category Cat_i that can be found in that particular cluster. See Figure 5.8 for a general representation.

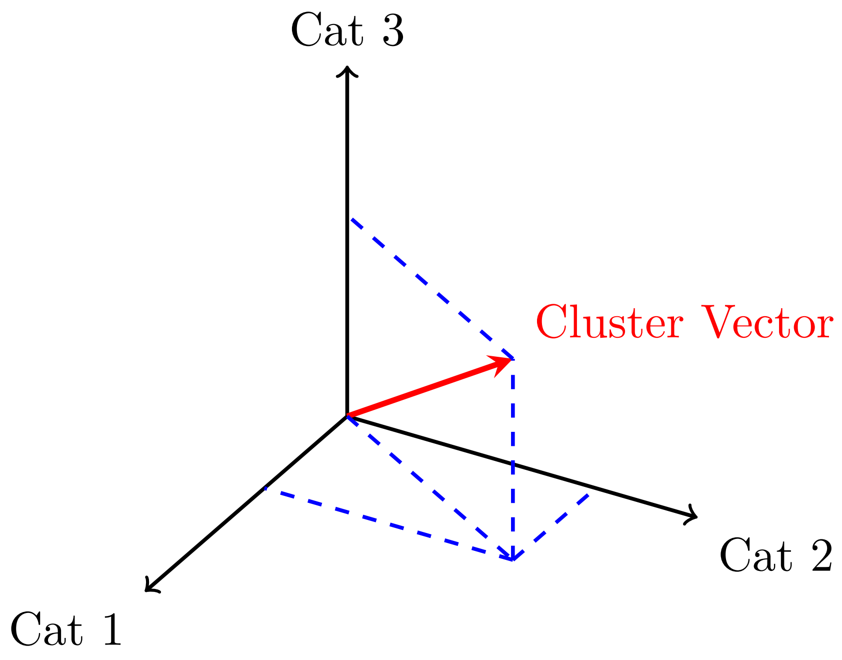


Figure 5.8.: An example of cluster vector for three categories

5.6.2. EMD Gaussians

To construct this particular measure, we shall first collect the average visit duration and standard deviation for the individual amenities. A cluster that contains one amenity A is represented as a Gaussian curve, i.e., normal probability distribution. The curve's center is at the average duration of the amenity A and its standard deviation is the one of the amenity. When n amenities A exist in the cluster, the representation will be an A curve multiplied n times. Multiple amenities, each appearing multiple times, will result in a curve that is the linear combination of the individual representations of the amenities as normal distribution curves. See Figure 5.9 for a visualization of the summing process.

$$EMD(\mathcal{C}_i) = \sum_{j=1}^{|amenities|} K_{ij} \times A_j \tag{5.1}$$

$$\forall i \in \{1, ..|clusters|\} \text{ and } \forall j \in \{1, ..|amenities|\}$$

where A_j is an amenity that appears K_{ij} times in the cluster $\mathcal{C}_i$.

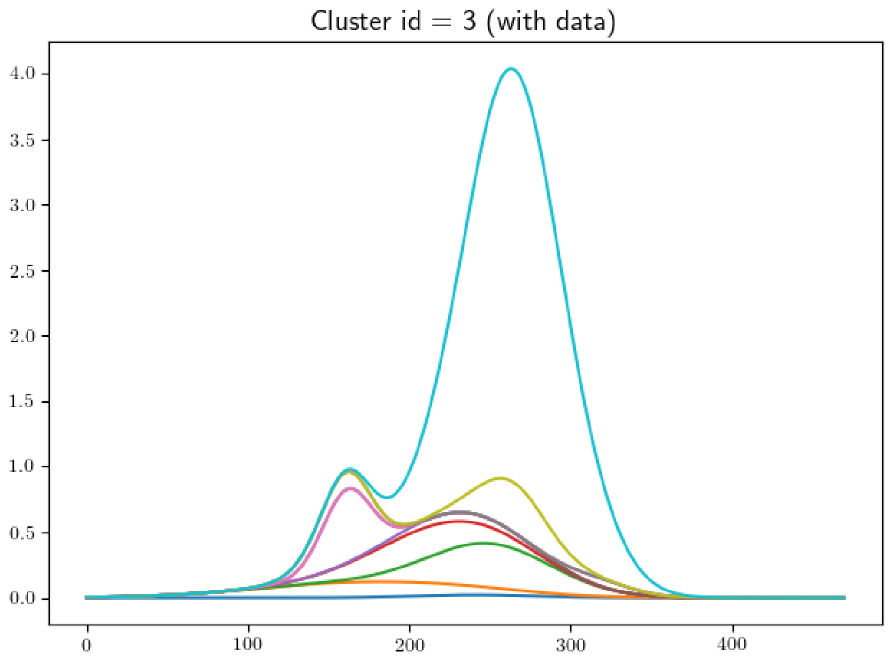


Figure 5.9.: The summing of Gaussians resulting in an EMD Gaussian

5.7. Similarity Functions

The option of transfering parking profiles involves a similarity function that quantifies how “close” two parking configuration are. Having defined urban measures for clusters, we need functions that take pairs of such measures and return a similarity or distance value. The metrics that we define are *cosine similarity* and *EMD*.

5.7.1. Cosine Similarity

We introduced the mathematical cosine similarity in Section 3.3. In our use case, the cosine vectors are the urban measure vectors just defined.

In practice, the function is calculated in PostgreSQL, which takes all combinations of pairs of cluster and performs the computation for their vectors.

5.7.2. Earth Mover's Distance

The background on earth mover's distance was given in Section 3.4. With the EMD Gaussian curves between which the distance is measured having just been defined, recall that EMD is applicable only when the sum under both Gaussian curves is equal. Therefore, all EMD Gaussians will get normalized before EMD is computed.

Unlike cosine similarity, EMD is externalized in a Python script, as its computation is more involved. The code first calculates EMD Gaussians for each cluster, afterwards it determines the EMD distance for each pair of clusters. For each cluster, discrete Gaussians will be created for the bin representation, as explained in Section 3.4. The pair-wise EMD is afterwards computed using the *pyemd* Python library. The results are written in the *cluster_similarity* table.

5.8. Machine Learning Models

The prediction of parking occupancy is realized using machine learning. We choose to explore this methodology, following the solid results machine learning models have delivered for the various smart parking setting investigated (cf.Section 4.2). A machine learning model will be trained for every cluster *with* parking data.

The training data *features* are composed of the parking data enumerated in Section 5.1.

We shall aggregate feature values around the location unit id, as on the cluster level this is irrelevant. The occupancy rate is set as the target variable. The model training and evaluation is performed in Python via the *scikit-learn* libraries. Training makes use of grid search (cf. Section 3.2.7) to determine the best model for the training data. The error metric used is RMSE. Furthermore, a model is evaluated on the other clusters with parking occupancy data. The model will be persisted and saved on the system as a .pkl file, while the model metadata is saved in the database. On testing the models, the user can decide whether new models will be built, or if the persisted ones will be used.

5.9. Parking Occupancy Estimations

Once models have been built for the clusters *with* parking data, making estimations on parking occupancy in these areas is straightforward. We want to apply these models, however, on the clusters that are missing parking data. Based on the logic presented in Section 2.2, we derive the *estimation interval* for cluster $\mathcal{C}^j_{wout}$ based on the model of cluster $\mathcal{C}^i_{with}$ as:

$$EI(\mathcal{C}^i_{with}, \mathcal{C}^j_{wout}) = \mathcal{M}(\mathcal{C}^i_{with})(X) \star sim(\mathcal{C}^i_{with}, \mathcal{C}^j_{wout}) \tag{5.2}$$

$$\forall i \in \{0, ..., |\mathcal{C}_{with}| - 1\} \text{ and } \forall j \in \{0, ..., |\mathcal{C}_{wout}| - 1\}$$

.

X is a parking data record containing feature values. The result is an *estimation interval* that "stretches" the punctual estimation into an interval depending on the similarity measure value. The lower the similarity value is, the larger the length of the resulting estimation interval will be.

Notice that X should be valid for both $\mathcal{C}^i_{with}$ and $\mathcal{C}^j_{wout}$. Given that we have excluded the location unit id, specific to the training cluster, this is not a problem. The averaged values used for features related to the location unit id will be transferred as they are to the target cluster $\mathcal{C}^j_{wout}$.

Furthermore, we define an *estimation intersection interval*, whose purpose is to narrow down the computed estimation interval. An estimation intersection interval for the clusters $\mathcal{C}^i_{with}$ and $\mathcal{C}^j_{wout}$ is computed by intersecting the *estimation intervals* that have a better similarity among the clusters with data $\mathcal{C}^0_{with}, ..., \mathcal{C}^{i-1}_{with}$ and the same cluster without data $\mathcal{C}^j_{wout}$.

$$EII(\mathcal{C}^i_{with}, \mathcal{C}^j_{wout}) = \bigcap_{k=0}^{i-1} EI(\mathcal{C}^k_{with}, \mathcal{C}^j_{wout}) \tag{5.3}$$

$$where \begin{cases} sim(\mathcal{C}^k_{with}, \mathcal{C}^j_{wout}) < sim(\mathcal{C}^i_{with}, \mathcal{C}^j_{wout}), k \in \{0,..,i-1\} & \text{for EMD} \\ sim(\mathcal{C}^k_{with}, \mathcal{C}^j_{wout}) > sim(\mathcal{C}^i_{with}, \mathcal{C}^j_{wout}), k \in \{0,..,i-1\} & \text{for cosine} \end{cases} \tag{5.4}$$

$$\forall i \in \{0, ..., |\mathcal{C}_{with}| - 1\} \text{ and } \forall j \in \{0, ..., |\mathcal{C}_{wout}| - 1\}$$

.

5.10. Data Processing and Persistence

Here we shall give a brief account of the implementation.

The backend system consists of a PostgreSQL database, whose table overview can be seen in Figure 5.10. The database has PostGIS extensions, in order to natively perform GIS-specific operations.

The initialization starts by importing the parking data. The occupancy and blocks data are available as CSV files and are imported directly into the *blocks*, i.e. location data table, and respectively *occupancy* tables. Acquiring the city data from the downloaded OSM layer file is performed using the *osm2pgsql* tool. Only *points of interest* with a few attributes will be retained, however, among which the *amenity* column. The points of interest will be merged with the blocks for a specified *merge distance.*

The clustering process is started as a bash script that relies on a Python module to perform the clustering itself. For each urban measure there is table where information about cosine vectors, respectively EMD Gaussians are stored. *cluster_vectors* holds information on each cluster, amenity categories, and number of amenities that fall into the respective category. The table *cluster_gaussians* stores cluster id, amenity name, and the amenity count that is to be found in the respective cluster. The EMD computation has been externalized in Python, while the cosine similarity is performed directly in PostgreSQL. The table *cluster_similarities* stores pairs of cluster ids, together with boolean flags indicating parking data. The computed similarity values are stored together with their type of similarity function, e.g., cosine or EMD.

The process of building machine learning models is a bash script that first calls a Python module, then exports the model results and similarity values in JSON format.

In the respective Python module, the training data is acquired from the database and the models are built as described in Section 5.8 using the *scikit-learn* library. The models metadata are stored in the table *models*, which contains information such as source cluster id, training error, training timestamp, training duration, target cluster id, and testing error. The models themselves are persisted as .pkl files.

The visualization of the results happens in Leaflet, which accesses the exported JSON objects representing the clusters, similarity values, and model information. The JSONs are stored in Javascript files.

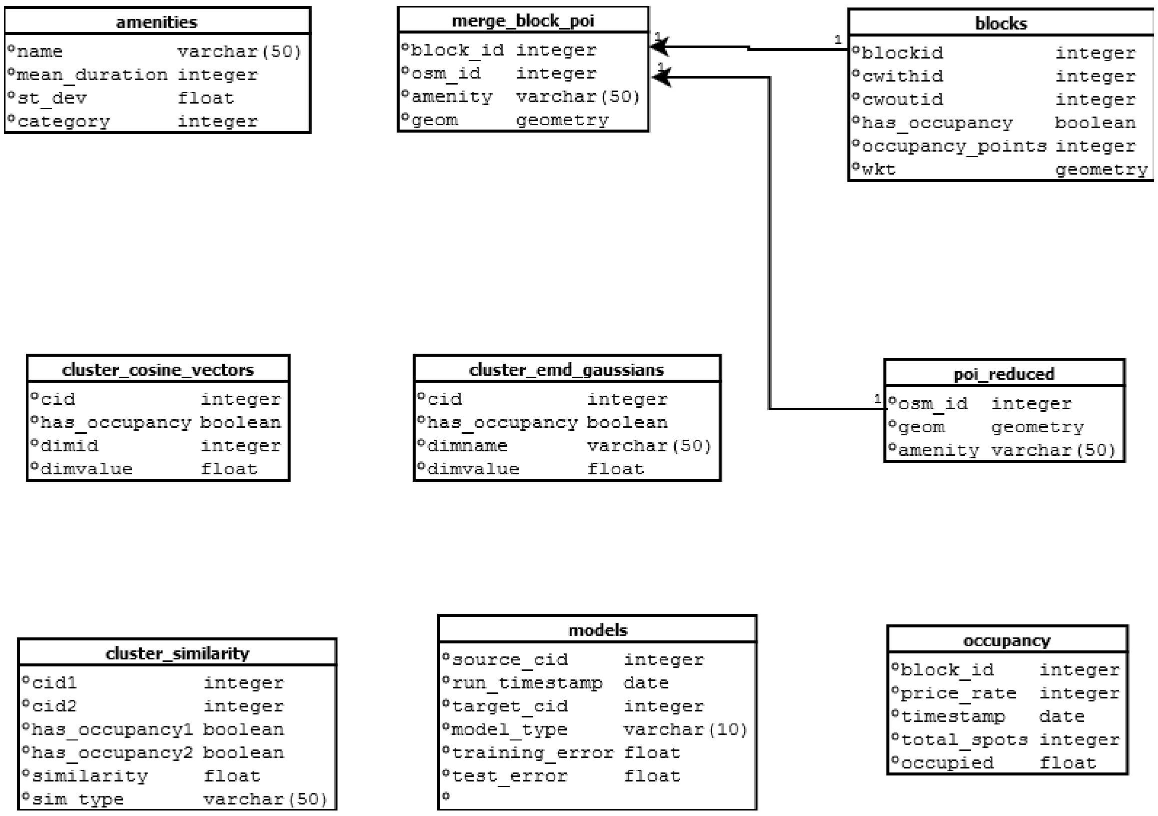


Figure 5.10.: A diagram of the database tables

6. Evaluation

This chapter discusses a concrete use case, on which the defined approach is applied. In Chapter 5, the proposed solution has been treated in a general manner, agnostic to the exact data location of parking and city data. Various parameters used throughout the process, such as *merge distance* or the number of clusters have been ignored up to now. In this chapter, we shall provide concrete inputs and set parameters in order to get actual results. To highlight particular artifacts and emphasize trends, the application is run repeatedly for different inputs and parameters. User-interface aspects shall be presented as well.

First, we take parking data from the SF*park* project and determine the datasets that are relevant for our purposes. Second, we instantiate the city data with OpenStreetMap data from San Francisco and obtain the available public amenity information. Google Places provides the visiting duration values for the amenities. Subsequently, we discuss relevant values for the *merge distance*. Remarks about the clustering process, urban measure, similarity values, as well as about the model training will follow.

The main evaluation will take place among the clusters *with* parking data, as estimation errors can be calculated in this case for the built models. Finally, concrete estimations for clusters *without* parking data will be shown.

6.1. SF*park* Parking Data

In Section 5.1, the types of parking data that our approach supports have been presented in detail. We shall instantiate this collection of data with the parking data used in the SF*park* project. An overview of the project has been presented in Section 3.11. The general form of the parking data presented already has been originally modeled after the SF*park* data, thanks to the its large diversity and detail. We will therefore find all types of parking data: occupancy, traffic, events, weather, parking revenue, and fuel price.

The actual SF*park* data has some particularities. While the *occupancy data* (cf. Section 5.1.1) is provided with reference to *blocks* as location units, all the other data sets use different location units. For the traffic and events datasets, the location units are street names. For parking revenue, it is districts. In case of weather and fuel price, the location reference is the whole city of San Francisco.

The SF*park* data can be visualized in Figure 6.1 using Leaflet.

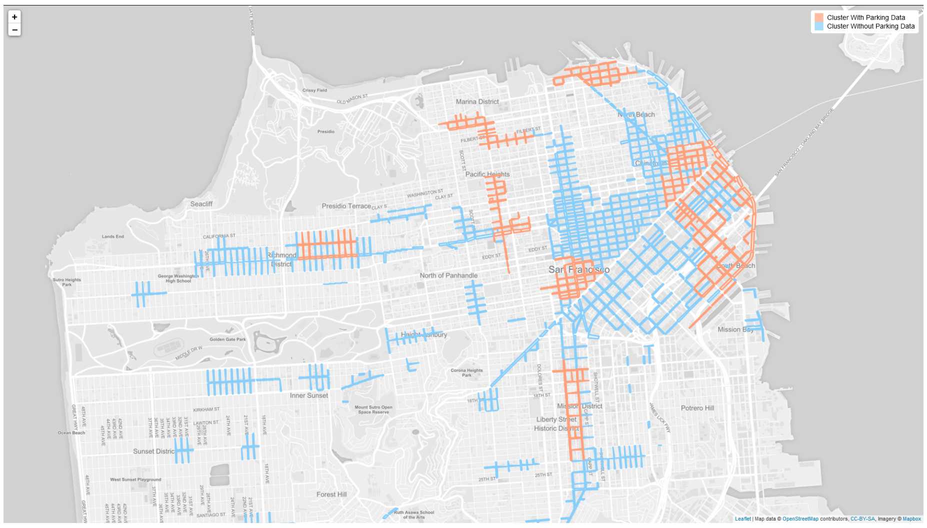


Figure 6.1.: The blocks accounted for in SF*park*. The light blue ones are blocks *without* parking data, the light red ones are *with* parking data.

6.2. OpenStreetMap for San Francisco

Following the selection of SF*park* data as parking data, the city data is found in the corresponding OpenStreetMap layer for San Francisco. The actual public amenity information, collected from POIs, is listed in Table 6.1.

arts_centre	dojo	marketplace	shelter
bank	embassy	music_rehearsal_place	shop
bar	fast_food	music_school	spa
biergarten	grocery	nightclub	stripclub
bureau_de_change	gym	pet_grooming_shop	studio
cafe	hookah_lounge	pharmacy	training
clinic	ice_cream	police	veterinary
clothes_store	karaoke	post_office	vintage_and_modern_resale
community_centre	lan_gaming_centre	pub	
dentist	laundry	restaurant	
doctors	library	salon	

Table 6.1.: List of all OSM amenities found in the SF*park* blocks

6.3. Merging Parking and City Data

We explained in Section 5.4 that the merging of parking and city data depends on the geometrical reference systems of both datasets. In case of SF*park*, the blocks are given in latitude and longitude. In OpenStreetMap, the geometry is set to EPSG 4326. With both systems using the same reference, we can therefore set a *merge distance*. The distance d should express the impact that a POI P has on the block B, when $dist(P, B) = d$, e.g., the parking demand that the restaurant induces on a parking block situated at d meters away. We assign to it distances of 100m, 200m, and 400m. Note, that in the implementation, we have used a point distance function that calculates distances in terms of latitude and longitude degrees. Given that we are using EPSG 4326, the coordinate distance takes into account the absolute latitude and longitude values for San Francisco.

6.4. Clustering

We will apply K-Means to cluster the city areas. In the evaluation, we will refer to the number of clusters *with* parking data as *the number of clusters*. The area without parking data is going to be split into a proportional number of clusters, as the sizes of clusters should be kept roughly equal for both sides. It turns out that the proportion is approximately 2.6, following the division between the total number of blocks from each group (cf. Table 6.2). We have chosen two number of clusters to run the evaluation, namely 8 clusters and 16 clusters. The area without parking data will therefore have 20 and 41 clusters, respectively.

- ▷ total number of blocks in the clusters *with* parking data: 409
- ▷ total number of blocks in the clusters *without* parking data: 1054
- ▷ proportion of cluster sizes: $1054/409 = 2.58$

Table 6.2.: The number of clusters between the parking data divide

After running the K-Means clustering process, the Leaflet map shall reveal the individual clusters by highlighting them on mouseover. The clusters *with* parking data will turn dark red, while the clusters *without* parking data will appear in dark blue (cf. Figure 6.2 and Figure 6.3).

When hovering over any red cluster, we see that cluster turning dark red. Additionally, another red cluster turns dark red. This is its most similar cluster *with* parking data based on the current similarity value. When hovering over a blue cluster, it turns dark blue. Additionally, three red clusters will turn dark red. These clusters *with* parking data are the most similar to the highlighted blue cluster. The cosine similarity values are used here.

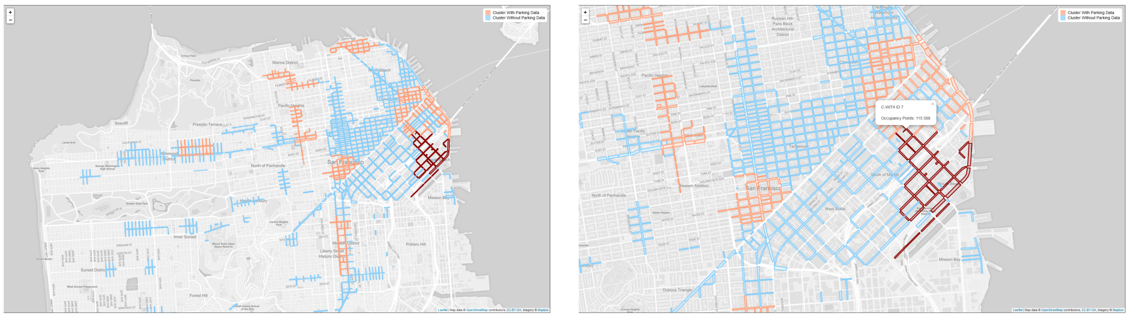


Figure 6.2.: Highlighted cluster *with* parking data and its pop-up information

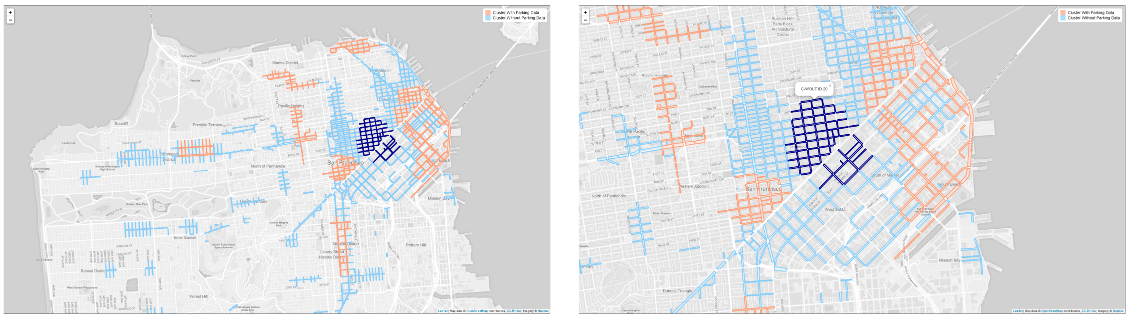


Figure 6.3.: Highlighted cluster *without* parking data and its pop-up information

Following the clustering of block, the resulting number of aggregated blocks per cluster is shown in Table 6.3. This is worth taking into account when training the machine learning models, as these will average over pieces of information contained in individual blocks.

Minimum:	8	Minimum:	8
Maximum:	86	Maximum:	116
Mean:	40.9	Mean:	44.8
St. deviation:	19.7	St. deviation:	24.3

Table 6.3.: Number of blocks that have been clustered.
Left: clusters *with* parking data. Right: clusters *without* parking data.

6.4.1. Urban Measure

Before computing *cosine vectors* and *EMD Gaussians*, we will establish the visit duration in every amenity. For this, we use information gathered from Google Places (cf. Section 3.9).

We collected information from 470 places in San Francisco, for which a maximum duration of stay was provided (the minimum duration is not always given, as indicated earlier). The data was obtained by manually navigating to every business place and writing the duration visit information in a spreadsheet. This piece of information is not accessible yet via the Google Places API[1]. The results are shown in Table 6.4. The numbers have been rounded to the nearest integer. We have included only amenities for which at least two stay duration sources were found.

amenity name	**mean**	**stdev**	**cat**	**amenity name**	**mean**	**stdev**	**cat**
arts_centre	110	37	3	laundry	78	16	2
bank	42	65	2	library	83	13	2
bar	121	38	3	music_school	120	30	3
cafe	76	39	2	nightclub	189	20	3
clinic	100	29	3	pharmacy	25	20	1
clothes_store	41	37	2	post_office	16	2	1
community_centre	119	40	3	pub	135	21	3
dentist	104	35	3	restaurant	135	32	3
doctors	60	42	2	salon	141	53	3
embassy	75	24	2	shelter	90	0	2
fast_food	31	15	2	shop	43	21	2
grocery	20	10	1	spa	161	54	3
gym	100	22	3	stripclub	140	46	3
hookah_lounge	130	17	3	studio	60	0	2
ice_cream	23	7	1	veterinary	67	29	2
karaoke	188	15	3	vintage_modern_resale	38	32	2

Table 6.4.: All amenities listed with their corresponding mean visit duration and standard deviation, as collected from Google Places. The assigned category for cosine vectors is included.

Alongside this information, we need the amenity categories in order to derive the *cosine vectors*. As defined in Section 5.6.1, the categories are based on the visit duration mean. We decided for the category values shown in Table 6.5. The assigned partitions for every amenity are shown in Table 6.4.

Thresholds on mean stay duration:	
Category 1:	0 to 30 minutes
Category 2:	31 to 90 minutes
Category 3:	above 91 minutes

Table 6.5.: The amenity categories

The calculation of *EMD Gaussians* relies on both the mean and standard deviation of the amenity visit duration, as defined in Section 5.6.2. For 8 clusters, the Gaussians produced for the clusters with parking data are shown in Figure 6.4.

[1]Google Feature Request: `https://issuetracker.google.com/issues/35827350`

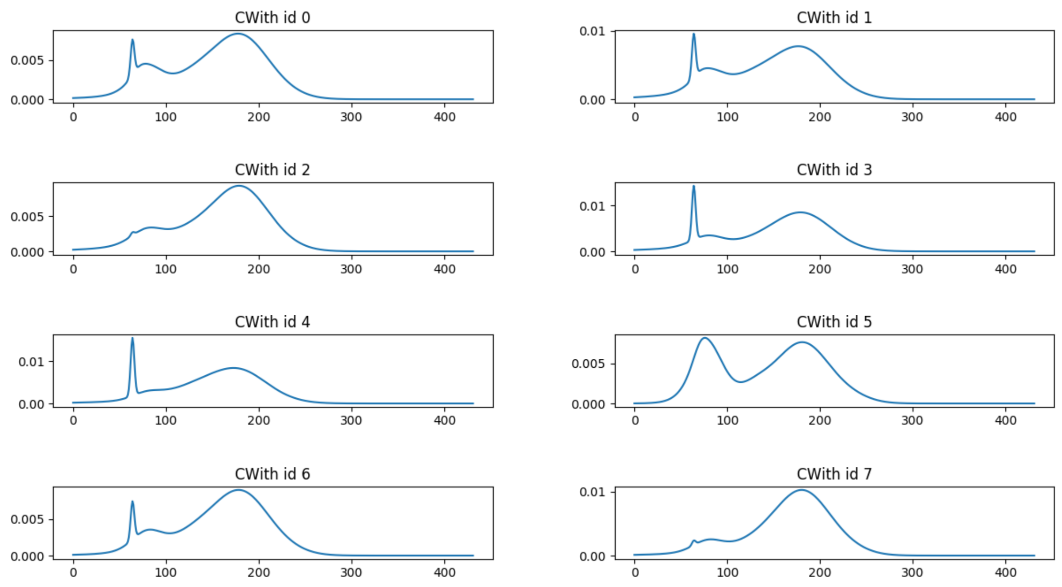


Figure 6.4.: Example of EMD Gaussians for clusters with parking data for 8 clusters.

6.5. Model Training

We use four methods to train models for the clusters *with* parking data: *decision trees*, *support vector machines*, *multilayer perceptrons*, *gradient boosted trees*. As for training data, the SF*park occupancy data* is used in its integrity, following the general description in Section 5.1.1 and having *blocks* as location unit. It turns out that training on the additional SF*park* data, traffic and events, encounters some problems.

The *traffic data* do not share the same location unit with the main occupancy data, i.e., block. When aggregating traffic data on the district level, which is available for the occupancy data as well, does not provide an additional value to the training. Indeed, using *scikit-learn*'s feature selection method to rank feature impact on training [11], the traffic's features (average road occupancy, the average vehicle count, the median speed, and the average speed of cars) ranks significantly lower than the occupancy features.

The SF*park events data* encounters the same problem as the traffic data: the location unit does not match the block. In fact, the events are marked for streets, whose association to blocks is not determinable. *Parking revenue data* is provided for districts, which again are too general to make a difference in training. Finally, *weather data* and *fuel data* are given per city, hence making even less an impact to improve the model.

As indicated in Section 5.8, the training data is averaged on all blocks so that it can be applied later on other clusters. The averaging is performed per timestamp, i.e., if multiple blocks have an occupancy record for the same time and block, the occupancy rate will be averaged for both of these. Features such as *price* and *parking capacity per block* are averaged as well. This means that the original collection of data records shrinks. In Table 6.6, the number of training records per cluster is shown, together with the number of occupancy records for that cluster. On average, there are 10.5 blocks that share the same datapoints.

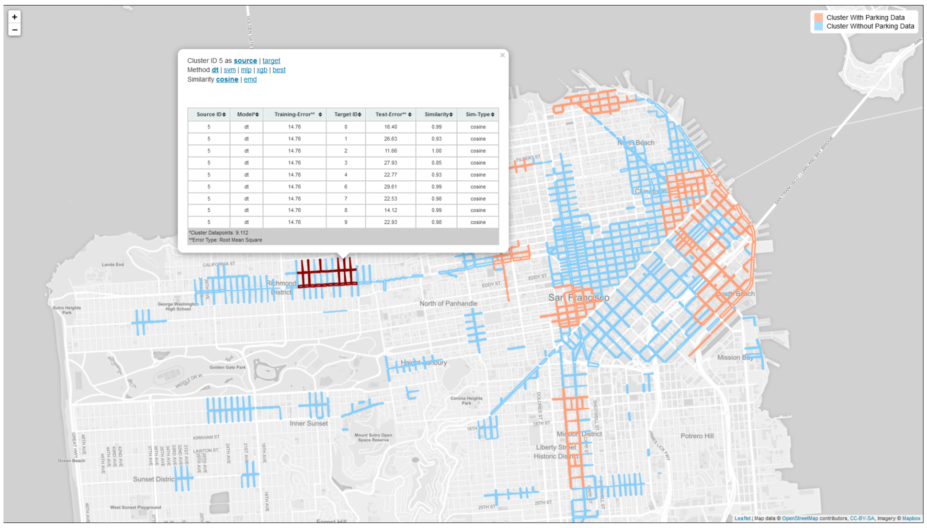
Cluster ID	**Data Points**	**Occupancy Points**	**Blocks per Timestamp**
0	9879	120 320	12.2
1	12 387	203 728	16.4
2	8713	61 839	7.1
3	6134	22 371	3.6
4	9586	110 463	11.5
5	9112	87 316	9.6
6	10 244	118 096	11.5
7	9500	115 588	12.2
8	9051	112 245	12.4
9	8713	76 230	8.7
Average	9332	102 820	10.5

Table 6.6.: Number of training points per model alongside the initial occupancy points within the containing blocks

6.6. Model Evaluation for Clusters *with* Parking Data

As foundation for this section, recall the background section that we have provided for understanding machine learning errors (cf. Section 3.2.6). In the following, we shall use *root mean square* to evaluate both the training and the testing errors of the built models. We investigate the built models among the clusters *with* parking data.

To visualize the training and testing results, we will use Leaflet (cf. Section 3.10). By clicking on a cluster *with* parking data, a pop-up will be shown containing a table with information on the models. The user can opt to see the model of the clicked cluster which is applied to all the other clusters, i.e., *source mode*, or one can see the results of all the other clusters'models being tested on the clicked cluster, i.e., *target mode*. In each mode, shown are both the training and test errors of the respective models. See Figure 6.5 for an example.

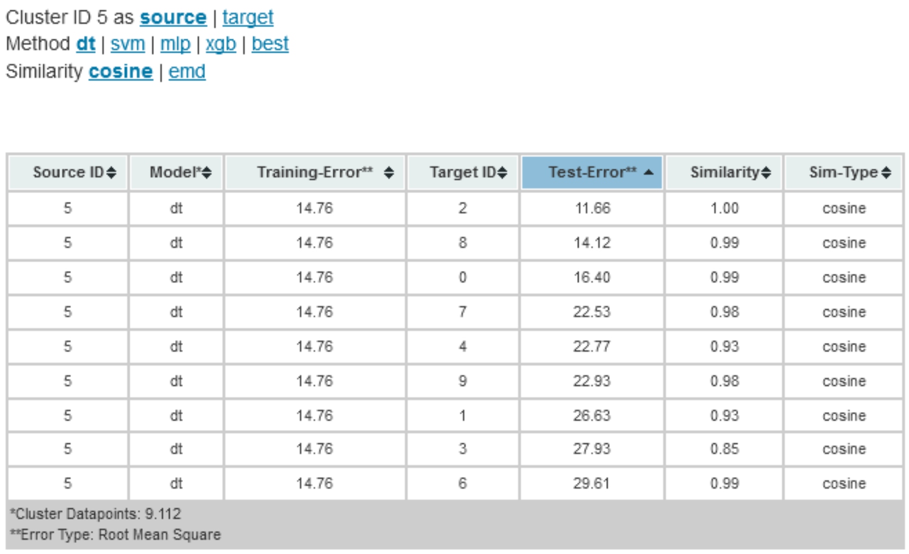


Cluster ID 5 as **source** | target
Method **dt** | svm | mlp | xgb | best
Similarity **cosine** | emd

Source ID	Model*	Training-Error**	Target ID	Test-Error**	Similarity	Sim-Type
5	dt	14.76	0	16.40	0.99	cosine
5	dt	14.76	1	26.63	0.93	cosine
5	dt	14.76	2	11.66	1.00	cosine
5	dt	14.76	3	27.93	0.85	cosine
5	dt	14.76	4	22.77	0.93	cosine
5	dt	14.76	6	29.61	0.99	cosine
5	dt	14.76	7	22.53	0.98	cosine
5	dt	14.76	8	14.12	0.99	cosine
5	dt	14.76	9	22.93	0.98	cosine

*Cluster Datapoints: 9.112
**Error Type: Root Mean Square

Figure 6.5.: Selected cluster *with* parking data and the pop-up table in Leaflet



Cluster ID 5 as **source** | target
Method **dt** | svm | mlp | xgb | best
Similarity **cosine** | emd

Source ID	Model*	Training-Error**	Target ID	Test-Error**	Similarity	Sim-Type
5	dt	14.76	2	11.66	1.00	cosine
5	dt	14.76	8	14.12	0.99	cosine
5	dt	14.76	0	16.40	0.99	cosine
5	dt	14.76	7	22.53	0.98	cosine
5	dt	14.76	4	22.77	0.93	cosine
5	dt	14.76	9	22.93	0.98	cosine
5	dt	14.76	1	26.63	0.93	cosine
5	dt	14.76	3	27.93	0.85	cosine
5	dt	14.76	6	29.61	0.99	cosine

*Cluster Datapoints: 9.112
**Error Type: Root Mean Square

Figure 6.6.: The pop-up table for the previous Leaflet view (cf. Figure 6.5)

6.6.1. Clustered- vs. Total Models

One assumption of our approach is that models originating from smaller clusters are better at predicting occupancy than models trained with the entire city area. We have compared the two types of models during our tests.

For each target cluster $\mathcal{C}_{target}$, we shall determine the source cluster $\mathcal{C}_{source}$ whose model has the best estimation error when applied on $\mathcal{C}_{target}$. Also, we will train a model containing the entire city area with parking data $\mathcal{A}$ minus $\mathcal{C}_{target}$'s data and compute this model's estimation error on $\mathcal{C}_{target}$. As it can be seen in Table 6.7 in the case of 8 clusters, and in Table 6.8 for 16 clusters, the cluster's models estimations are superior to the ones of the total model with very few exceptions.

$\mathcal{C}_{target}$	$\mathcal{M}(\mathcal{A} \setminus \mathcal{C}_{target})$	$error_{total}$	$error_{cluster}$	$\mathcal{M}(\mathcal{C}_{source})$	$\mathcal{C}_{source}$
0	xgb	18.20	16.10	xgb	6
1	dt	17.44	12.14	xgb	7
2	xgb	20.33	17.45	svm	5
3	xgb	17.59	13.35	xgb	1
4	dt	17.49	17.30	xgb	6
5	xgb	18.44	16.08	xgb	2
6	xgb	16.00	15.92	svm	0
7	dt	16.38	10.87	xgb	3

Table 6.7.: Comparison between the estimations of the total models versus the ones of clustered models, in case the city are with parking data is split into 8 clusters. Errors are expressed as RMSE.

$\mathcal{C}_{target}$	$\mathcal{M}(\mathcal{A} \setminus \mathcal{C}_{target})$	$error_{total}$	$error_{cluster}$	$\mathcal{M}(\mathcal{C}_{source})$	$\mathcal{C}_{source}$
0	mlp	19.53	13.40	xgb	15
1	xgb	17.82	17.18	xgb	7
2	xgb	20.79	17.97	svm	6
3	xgb	16.44	13.25	xgb	0
4	xgb	17.58	16.40	xgb	8
5	dt	17.59	10.93	xgb	0
6	xgb	18.44	16.29	xgb	2
7	xgb	16.73	16.38	xgb	9
8	xgb	13.57	14.14	xgb	10
9	xgb	16.90	15.95	xgb	10
10	xgb	16.25	16.45	xgb	7
11	mlp	21.73	14.95	xgb	11
12	xgb	20.33	15.42	xgb	0
13	dt	14.52	11.39	xgb	15
14	dt	22.93	18.73	svm	6
15	dt	20.63	13.33	xgb	0

Table 6.8.: Comparison between the estimations of the total model versus the ones of clustered models. In case the city are with parking data is split into 16 clusters. Errors are expressed as RMSE.

6.6.2. Best Model Method

Models were trained using four methods: decision trees, support vector machines, multilayer perceptrons, gradient boosted trees. We shall now find out, which method yields the best estimation errors, when the corresponding models are applied on clusters *with* parking data. Table 6.9 shows the distribution of best machine learning methods in case of 8 and 16 clusters. The values were obtained by summing up the number of times a method produced the least estimation error, i.e., RMSE, among the four methods for all combinations of clusters with parking data $(\mathcal{C}_{source}, \mathcal{C}_{target})$. Intermediate results can be seen in Appendix A. Extreme gradient boosting claims the first spot in both cases.

	dt	**svm**	**mlp**	**xgb**
8 clusters	24.6%	17.5%	12.3%	45.6%
16 clusters	14.6%	13.8%	13.8%	57.9%

Table 6.9.: The proportion of best models by means of machine learning method

6.6.3. Similarity Values vs. Estimation Errors

Recall that the goal of our approach was to replace occupancy estimations for clusters where no parking data is available with estimations based on cluster similarity values. Among clusters *with* parking data, the real occupancy values are known. This enables us to compute estimation errors for cluster models, which can then be correlated with the similarity values between clusters.

We shall use two correlation coefficients: the Pearson correlation coefficient and Spearman's rank correlation coefficient, which were introduced in Section 3.6.

We have evaluated both cosine and EMD similarity values in configurations of 8 and respectively 16 clusters. Additionally, we varied the *merge distance* to see how the correlation behaves. The similarity values are hence calculated for 100m, 200m, and 400m merge distance respectively. In Table 6.10 the final results are shown. For each correlation measure, the percent of similarity values were calculated that correlated positively, in case of EMD distance, or negatively, for cosine similarity, with the estimation errors. The models taken were trained with gradient boosted trees. For intermediate results, see Appendix A.

We notice that the cosine similarity achieves better results than EMD for the same testing configuration, peaking at 100% negative correlation for 8 clusters and 100m merge distance. Its average Pearson coefficient is -0.55, while the mean Spearman rank coefficient is -0.49. EMD positively correlates the most for the same testing configuration, when the average Pearson coefficient is at 0.28 and Spearman's rank coefficient equals 0.23. There is a clear descending trend in correlations, as the merge distance increases. Also, the results for 8 clusters are superior to the ones when the city is split in 16 clusters.

	8 clusters				**16 clusters**			
	cosine	rank_cosine	emd	rank_emd	cosine	rank_cosine	emd	rank_emd
100m	100%	100%	87.5%	75%	75%	75%	68.75%	62.5%
200m	75%	75%	75%	75%	75%	75%	56.3%	56.3%
400m	62.5%	50%	75%	75%	68.8%	68.8%	62.5%	62.5%

Table 6.10.: Correlations between similarity values and model estimations errors for pairs of clusters *with* parking data ($\mathcal{C}_{source}, \mathcal{C}_{target}$). For cosine similarity, the proportions show the negative correlation, while for EMD, they express the positive correlation. The Pearson coefficient and Spearman's rank coefficient were used.

6.7. Estimations for Clusters *without* Parking Data

Following the procedure in Section 5.9, we apply the models trained on SF*park* data on clusters *without* parking data. The testing data records are composed of values equal to the averages of the respective data types in all clusters *with* parking data. This is the case for *parking price* and *parking capacity*. One piece of data that still needs to be provided so that the estimation is computed, however, is the date and time. For convenience, we choose the next day at the point when the user starts the model training. On the Leaflet map, the time is offered as a drop-down list inside the pop-up for a cluster without data. The user can choose between eight different times throughout the day for which the estimations are calculated.

The estimation is displayed as an *estimation interval*, as defined in Section 5.9. The *estimation intersection interval* is displayed in a separate column. In Leaflet, the estimations can be visualized as in Figure 6.7 and Figure 6.8.

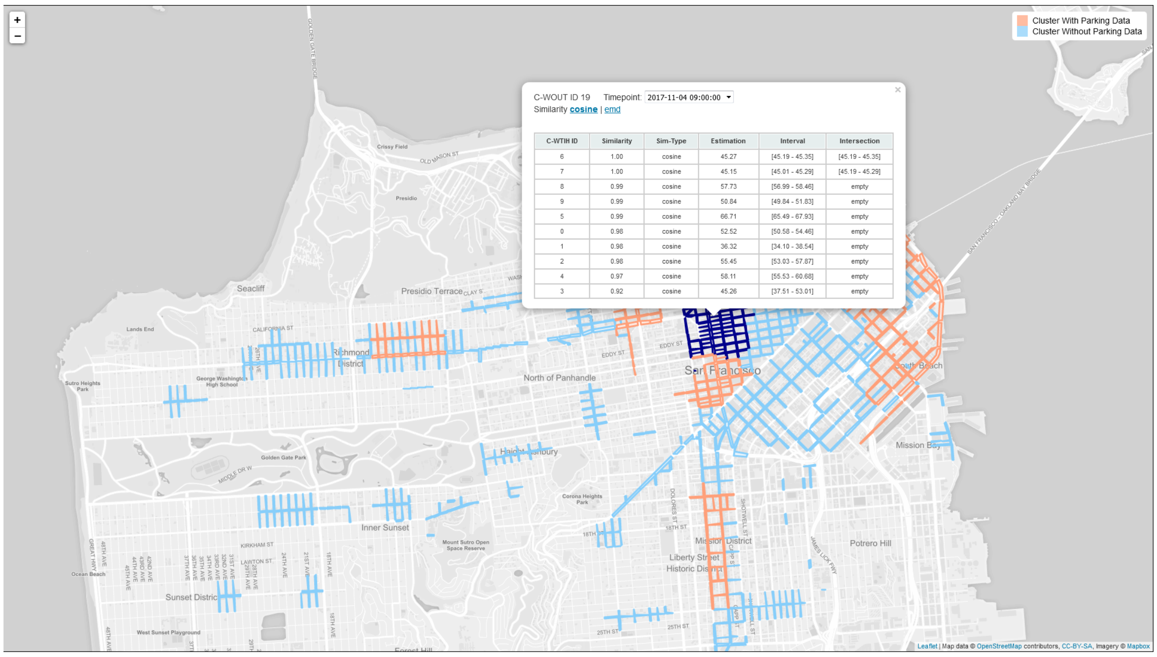
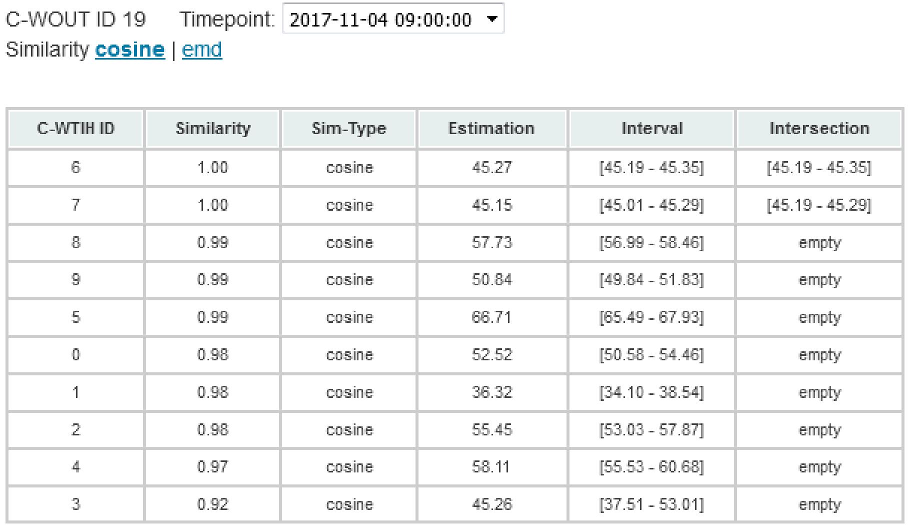


Figure 6.7.: Selected cluster *without* data and its table pop-up in Leaflet


C-WOUT ID 19 Timepoint: 2017-11-04 09:00:00
Similarity **cosine** | emd

C-WTIH ID	Similarity	Sim-Type	Estimation	Interval	Intersection
6	1.00	cosine	45.27	[45.19 - 45.35]	[45.19 - 45.35]
7	1.00	cosine	45.15	[45.01 - 45.29]	[45.19 - 45.29]
8	0.99	cosine	57.73	[56.99 - 58.46]	empty
9	0.99	cosine	50.84	[49.84 - 51.83]	empty
5	0.99	cosine	66.71	[65.49 - 67.93]	empty
0	0.98	cosine	52.52	[50.58 - 54.46]	empty
1	0.98	cosine	36.32	[34.10 - 38.54]	empty
2	0.98	cosine	55.45	[53.03 - 57.87]	empty
4	0.97	cosine	58.11	[55.53 - 60.68]	empty
3	0.92	cosine	45.26	[37.51 - 53.01]	empty

Figure 6.8.: The pop-up table of a cluster without data. Notice the drop-down list from which the time can be selected.

7. Conclusion & Future Work

In this work, we have presented an attempt at approximating street parking occupancy in cities. Under the assumption that parking data is lacking, in order to build scalable occupancy prediction systems, we proposed an alternative solution to the ones previously developed for this problem.

We built parking profiles by using complementary city data, which localize various types of public amenities and indicate the average visit duration there. All data has been made available in an established RDF format, so that it can be easily reused. We merged the *parking data* with the *city data* by matching parking *location units* to *points of interest*, split the city into clustered areas, and built machine learning models for them. K-Means was used to cluster the location units, while four methods were employed to train models for the clusters: decision trees, support vector machines, multilayer perceptrons and extreme gradient boosting. Based on the city data, urban measures were built in the form of *cosine vectors* and *EMD Gaussians*, both of which took advantage of the mean visit duration and its standard deviation. The vectors were part of the *cosine similarity* computation, while Gaussians contributed to the *earth mover's distance* calculation. The occupancy estimations for clusters *without* parking data were defined in terms of model estimations from clusters *with* parking data and the corresponding cluster similarity values. The estimations are expressed as intervals which extend the model prediction values by the magnitude of the similarity values.

As use case, we chose the SF*park* project from San Francisco, which gathered parking data for more than 2 years starting in 2011 and now offers it for free usage. The city data was collected from OpenStreetMap as amenity information, and from Google Business as stay duration values. Both sources are open and free of charge. Over 30 types of public amenities were found in the San Francisco blocks, which corroborated with over 470 Google Places sources, lead to building the urban measures and similarity values.

The results confirmed that clustering the city into smaller areas yields better occupancy estimations than those of entire city area models. Following our tests, the best machine learning model turned out to be *extreme gradient boosting*. We used the clusters *with* parking data for the evaluation of the similarity values and calculated correlation coefficients between the similarity values and the estimation errors, using both absolute values and ranks. The best correlation were reached for the 100m merge distance for 8 clusters, averaging at -0.55 as Pearson Coefficient and -0.49 as Spearman's rank coefficient. In the same configuration, both *cosine similarity* and *EMD distance* reached their best results from all the test configurations. Overall, *cosine similarity* achieved better correlations than *emd*. Finally, the models for 8 clusters produced superior results over the models for 16 clusters.

7.1. Future Work

To further investigate parking occupancy prediction given the assumptions in this work, there are several improvements or alternative approaches that can be undertaken.

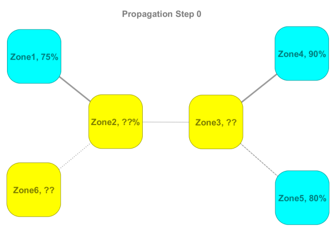
1. **Use more parking data**. This point makes sense for any kind of system that estimates parking occupancy, as machine learning models return better estimations when the parking data *is relevant*, i.e. not too specific so that it leads to overfitting. In the present work, several pieces of data could not be integrated because of merging issues, i.e., the location unit did not coincide with the occupancy data's block. Traffic, events, weather, etc. could improve estimation results and hence the final estimations for clusters without parking data. Other sources for parking occupancy data can be found for the cities of Cologne [22], Zurich [37], Santa Monica [30]. In Germany, Deutsche Bahn provides an API to data from parking around train stations [6]. Data pertaining to street occupancy is however hard to find. At the time of writing, open data portals mostly provide the location of parking lots, parking meters, parking price and opening times, if applicable.

2. **Use more city data**. The parking profiles in the present work are relying on the public amenities from OpenStreetMap. OSM has great potential as a collaborative map service but it lacks many pieces of information that are available in Google Maps. Data such as opening hours would be interesting to include into the urban measure. The parking profile could then take into account the number of public amenities that are available at a certain point in time. Furthermore, the stay duration data collected for the present approach is currently limited, as Google does not provide an API to access this data. Adding more stay duration data may fine tune the emerging similarity values. Overall, more and finer city data, together with an appropriate representation and similarity function could eventually improve the occupancy estimations for clusters without parking data.

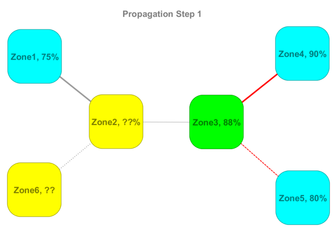
3. **Integrate city data in machine learning models**. An alternative to building urban measures and similarity functions is to let machine learning figure out the similarities by itself. One can add the city data as further training information for clusters. The models are then applied on clusters without parking

data and return occupancy estimations. The difference here is that models will be built for all clusters, including the ones without parking data. This also has the disadvantage of not using most of the parking data for training. The benefit of finding better similarities by leveraging unknown patterns in the city data may, however, outweigh this drawback.

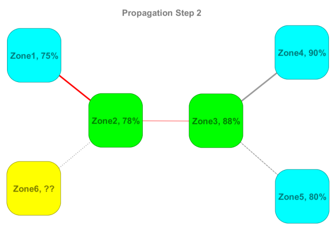
4. **Apply semi-supervised machine learning**. Another relevant machine learning approach in this case is based on organizing the city areas as an undirected graph. The vertices represent the clusters with their respective occupancy data, while the edges between them are assigned similarity values. Initially, only a part of the vertices have the occupancy value known, i.e. the clusters with parking data, while the rest has undetermined occupancy, i.e., the clusters without parking data. At each step, the value for a vertex whose value is undetermined is being computed by considering the occupancies of the linked vertices and their corresponding similarity values. See Figure 7.1 for an simplified example.



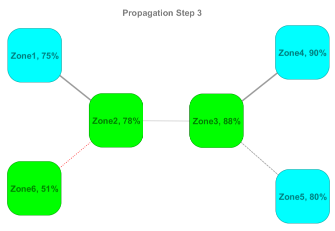
(a) Propagation Step 0



(b) Propagation Step 1



(c) Propagation Step 2



(d) Propagation Step 3

Figure 7.1.: Process of inferring occupancy values for untracked zones using occupancy values from tracked zones and their pair-wise similarity values as edges

A. Appendix

A.1. Evaluation

Cluster ID	**Cosine**		**EMD**	
	Correlation	Correlation Rank	Correlation	Correlation Rank
0	-0.21	-0.03	0.12	0.21
1	-0.89	-0.76	0.66	0.53
2	-0.42	-0.49	0.42	0.46
3	-0.35	-0.38	-0.83	-0.64
4	-0.79	-0.81	0.59	0.43
5	-0.87	-0.73	0.71	0.54
6	-0.26	-0.10	0.07	0.00
7	-0.63	-0.63	0.48	0.32
Positive/Negative (abs)	8	8	7	6
Positive/Negative (%)	100.0	100.0	87.5	75.0
Mean correlation	-0.55	-0.49	0.28	0.23

Table A.1.: The correlations coefficients between similarity values and estimation errors for a configuration of 8 clusters and 100m merge distance.

Cluster ID	**Cosine**		**EMD**	
	Correlation	Correlation Rank	Correlation	Correlation Rank
0	-0.39	-0.22	0.40	0.36
1	-0.85	-0.79	0.64	0.60
2	-0.20	-0.29	0.41	0.39
3	-0.21	0.04	-0.82	-0.83
4	0.39	0.44	0.64	0.68
5	-0.93	-0.91	0.70	0.61
6	0.01	-0.15	-0.35	-0.29
7	-0.55	-0.56	0.49	0.36
Positive/Negative (abs)	6	6	6	6
Positive/Negative (%)	75.0	75.0	75.0	75.0
Mean correlation	-0.34	-0.30	0.26	0.23

Table A.2.: The correlations coefficients between similarity values and estimation errors for a configuration of 8 clusters and 200m merge distance.

Cluster ID	**Cosine**		**EMD**	
	Correlation	Correlation Rank	Correlation	Correlation Rank
0	-0.53	0.00	0.58	0.57
1	-0.71	-0.68	0.57	0.67
2	0.39	0.43	0.27	0.25
3	0.00	0.41	-0.73	-0.79
4	0.37	0.51	0.58	0.71
5	-0.71	-0.65	0.67	0.64
6	-0.09	0.04	-0.39	-0.29
7	-0.58	-0.69	0.43	0.36
Positive/Negative (abs)	5	4	6	6
Positive/Negative (%)	62.5	50.0	75.0	75.0
Mean correlation	-0.23	-0.08	0.25	0.27

Table A.3.: The correlations coefficients between similarity values and estimation errors for a configuration of 8 clusters and 400m merge distance.

Cluster ID	**Cosine**		**EMD**	
	Correlation	Correlation Rank	Correlation	Correlation Rank
0	-0.34	-0.43	0.36	0.36
1	-0.41	-0.49	0.19	0.25
2	-0.39	-0.40	0.30	0.43
3	-0.43	-0.36	0.33	0.35
4	-0.26	-0.17	0.32	0.34
5	-0.31	-0.27	0.26	0.16
6	-0.34	-0.24	-0.25	-0.08
7	0.30	0.45	-0.10	-0.21
8	-0.40	-0.46	-0.24	-0.11
9	-0.11	-0.18	0.30	0.37
10	0.15	0.04	-0.15	-0.19
11	-0.57	-0.44	0.30	0.30
12	0.53	0.67	-0.51	-0.50
13	-0.22	-0.19	0.06	-0.10
14	0.27	0.30	0.08	0.10
15	-0.66	-0.59	0.41	0.36
Positive/Negative (abs)	12	12	11	10
Positive/Negative (%)	75.0	75.0	68.8	62.5
Mean correlation	-0.20	-0.17	0.10	0.11

Table A.4.: The correlations coefficients between similarity values and estimation errors for a configuration of 16 clusters and 100m merge distance.

Cluster ID	**Cosine**		**EMD**	
	Correlation	Correlation Rank	Correlation	Correlation Rank
0	-0.33	-0.31	0.16	0.30
1	-0.30	-0.41	0.08	0.20
2	-0.27	-0.32	0.26	0.29
3	-0.45	-0.37	0.28	0.29
4	-0.33	-0.30	0.25	0.30
5	0.42	0.33	0.16	0.05
6	-0.38	-0.39	-0.20	-0.06
7	0.54	0.69	-0.25	-0.32
8	-0.27	-0.28	-0.35	-0.35
9	-0.24	-0.32	0.30	0.33
10	-0.04	-0.14	-0.03	-0.19
11	-0.55	-0.49	0.26	0.31
12	0.60	0.71	-0.48	-0.42
13	-0.30	-0.22	-0.03	-0.16
14	0.47	0.58	-0.41	-0.46
15	-0.58	-0.48	0.35	0.20
Positive/Negative (abs)	12	12	9	9
Positive/Negative (%)	75.0	75.0	56.3	56.3
Mean correlation	-0.13	-0.11	0.02	0.02

Table A.5.: The correlations coefficients between similarity values and estimation errors for a configuration of 16 clusters and 200m merge distance.

Cluster ID	**Cosine**		**EMD**	
	Correlation	Correlation Rank	Correlation	Correlation Rank
0	-0.47	-0.58	0.38	0.38
1	-0.02	-0.15	-0.23	-0.04
2	0.42	0.47	0.07	0.25
3	-0.53	-0.52	0.35	0.37
4	-0.42	-0.31	0.13	-0.01
5	0.31	0.24	0.30	0.30
6	-0.15	-0.19	-0.24	-0.01
7	0.02	0.19	-0.05	0.05
8	-0.32	-0.53	-0.16	0.10
9	-0.43	-0.47	0.33	0.36
10	-0.44	-0.16	0.25	0.02
11	-0.61	-0.55	0.29	0.20
12	0.46	0.43	-0.48	-0.37
13	-0.45	-0.43	0.07	-0.05
14	0.45	0.33	-0.06	-0.20
15	-0.55	-0.51	0.38	0.38
Positive/Negative (abs)	11	11	10	10
Positive/Negative (%)	68.8	68.8	62.5	62.5
Mean correlation	-0.17	-0.17	0.08	0.11

Table A.6.: The correlations coefficients between similarity values and estimation errors for a configuration of 16 clusters and 400m merge distance.

$\mathcal{C}_s$ / $\mathcal{C}_t$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0		dt	dt	dt	dt	dt	dt	xgb
1	svm		dt	svm	xgb	svm	dt	dt
2	xgb	mlp		xgb	xgb	xgb	mlp	xgb
3	xgb	xgb	xgb		mlp	xgb	xgb	xgb
4	svm	svm	svm	dt		dt	svm	xgb
5	xgb	mlp	xgb	svm	dt		svm	xgb
6	dt	xgb	xgb	xgb	dt	xgb		xgb
7	mlp	mlp	svm	xgb	xgb	mlp	xgb	

Table A.7.: Best machine learning method among the four (decision trees, support vector machines, multilayer perceptrons, extreme gradient boosting) for a 8 cluster test configuration. A model is trained on cluster's $\mathcal{C}_s$ data and applied on cluster $\mathcal{C}_t$.

	dt	**svm**	**mlp**	**xgb**
Absolute	14	10	7	26
Percent	24.6	17.5	12.3	45.6

Table A.8.: Overall number of machine learning methods that produced best estimations on the cluster pair level $(\mathcal{C}_s, \mathcal{C}_t)$ for an 8 cluster test configuration.

$\mathcal{C}_t$	**dt**	**svm**	**mlp**	**xgb**
0	4	3	2	6
1	3	0	0	12
2	1	0	1	13
3	2	1	1	11
4	3	1	1	10
5	6	0	0	9
6	1	3	3	8
7	1	3	6	5
8	2	6	2	5
9	2	3	3	7
10	2	3	1	9
11	3	5	3	4
12	1	1	4	9
13	2	0	4	9
14	0	3	1	11
15	2	1	1	11
Absolute	35	33	33	139
Percent	14.6	13.8	13.8	57.9

Table A.9.: The number of best machine learning methods whose models achieved best estimation errors for a 16 cluster test configuration. A model is trained on cluster's $\mathcal{C}_s$ data and applied on cluster $\mathcal{C}_t$. Due to space limitations, the $\mathcal{C}_s$ dimension was summed up.

Bibliography

[1] *Apache Jena.* https://jena.apache.org/

[2] *City Pulse.* http://www.ict-citypulse.eu/page/

[3] *Correlation and dependence - Wikipedia, the free encyclopedia.* https://en.wikipedia.org/wiki/Correlation_and_dependence

[4] *Cosine similarity - Wikipedia, the free encyclopedia.* https://en.wikipedia.org/wiki/Cosine_similarity

[5] *Cross-validation (statistics) - Wikipedia, the free encyclopedia.* https://en.wikipedia.org/wiki/Cross-validation_(statistics)

[6] *DB - Parkplätze API.* http://data.deutschebahn.com/dataset/api-parkplatz

[7] *Decision Tree Regression with AdaBoost.* http://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_regression.html

[8] *Decision Trees.* http://scikit-learn.org/stable/modules/tree.html

[9] *Demonstration of k-means assumptions.* http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html

[10] *EPSG:4326.* https://epsg.io/4326

[11] *Feature Seleciton.* http://scikit-learn.org/stable/modules/feature_selection.html

[12] *Free parking or free markets.* https://www.accessmagazine.org/spring-2011/free-parking-free-markets/

[13] *Google - Popular times, wait times, and visit duration.* https://support.google.com/business/answer/6263531?hl=en

[14] *Google - Visit Duration Image.* https://8ms.com/wp-content/uploads/2016/11/Popular-Times-Live.png

[15] *Google My Business.* https://www.google.com/business/

[16] *Leaflet - an open-source JavaScript library for mobile-friendly interactive maps.* http://leafletjs.com/

[17] *Machine Learning - Wikipedia, the free encyclopedia.* https://en.wikipedia.org/wiki/Machine_learning

[18] *Mean absolute error.* http://scikit-learn.org/stable/modules/model_evaluation.html#mean-absolute-error

[19] *Mean squared error.* http://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error

[20] *Mean squared error.* http://blog.minitab.com/blog/adventures-in-statistics-2/regression-analysis-how-to-interpret-s-the-standard-e

[21] *The number of cars worldwide is set to double by 2040.* https://www.weforum.org/agenda/2016/04/the-number-of-cars-worldwide-is-set-to-double-by-2040

[22] *Öffene Daten Köln.* https://www.offenedaten-koeln.de/dataset/taxonomy/term/52/field_tags/Transport%20und%20Verkehr-52?query=park&sorting=changed%7CDESC

[23] *Open Street Map.* https://www.openstreetmap.org/

[24] *Open Street Map POI Image.* http://wiki.openstreetmap.org/w/images/8/8a/Condado-pois.PNG

[25] *OpenStreetMap.* https://www.openstreetmap.org/

[26] *Osm2pgsql.* http://wiki.openstreetmap.org/wiki/Osm2pgsql

[27] *Pearson correlation coefficient - Wikipedia, the free encyclopedia.* https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient

[28] *R^2 score.* http://scikit-learn.org/stable/modules/model_evaluation.html#r2-score

[29] *Rank correlation - Wikipedia, the free encyclopedia.* https://en.wikipedia.org/wiki/Rank_correlation

[30] *Santa Monica - Open Data.* https://data.smgov.net/Transportation/Parking-Lot-Counts/ng8m-khuz

[31] *SFpark.* http://sfpark.org

[32] *SFpark - Open Data.* http://sfpark.org/how-it-works/open-data-page/

[33] *SFpark - Pilot Project Evaluation Summary.* http://sfpark.org/wp-content/uploads/2014/06/SFpark_Eval_Summary_2014.pdf

[34] *Spatial Reference System - Wikipedia, the free encyclopedia.* https://en.wikipedia.org/wiki/Spatial_reference_system

[35] *Spearman's rank correlation coefficient - Wikipedia, the free encyclopedia.* https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient

[36] *stackoverflow - world's largest developer community.* https://stackoverflow.com/

[37] *Stadt Zürich - Open Data.* https://data.stadt-zuerich.ch/dataset/parkleitsystem

[38] *SVM: Maximum margin separating hyperplane.* http://scikit-learn.org/0.18/auto_examples/svm/plot_separating_hyperplane.html

[39] *Wasserstein Metric - Wikipedia, the free encyclopedia.* https://en.wikipedia.org/wiki/Wasserstein_metric

[40] Berners-Lee, Tim: *Linked Data.* https://www.w3.org/DesignIssues/LinkedData.html. Version: 2006

[41] Brink, Henrik ; Richards, Joseph ; Fetherolf, Mark: *Real-world machine learning.* Manning Publications Co., 2016

[42] Caicedo, Felix ; Blazquez, Carola ; Miranda, Pablo: Prediction of parking space availability in real time. In: *Expert Systems with Applications* 39 (2012), Nr. 8, S. 7281–7290

[43] Caliskan, Murat ; Barthels, Andreas ; Scheuermann, Bjorn ; Mauve, Martin: Predicting parking lot occupancy in vehicular ad hoc networks. In: *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th* IEEE, 2007, S. 277–281

[44] Chen, Tianqi ; Guestrin, Carlos: Xgboost: A scalable tree boosting system.

In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* ACM, 2016, S. 785–794

[45] Chen, Xiao: Parking occupancy prediction and pattern analysis. In: *Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, Tech. Rep. CS229-2014* (2014)

[46] Chen, Zhirong ; Xia, Jianhong C. ; Irawan, Buntoro: Development of fuzzy logic forecast models for location-based parking finding services. In: *Mathematical Problems in Engineering* 2013 (2013)

[47] Group, RDF W.: *RDF 1.1 Concepts and Abstract Syntax.* `https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`. Version: 2014

[48] Group, RDF W.: *RDF 1.1 Primer.* `https://www.w3.org/TR/rdf11-primer/`. Version: 2014

[49] Group, RDF W.: *RDF 1.1 Semantics.* `https://www.w3.org/TR/2014/REC-rdf11-mt-20140225/`. Version: 2014

[50] Group, SPARQL W.: *SPARQL 1.1 Overview.* `https://www.w3.org/TR/sparql11-overview/`. Version: 2013

[51] Hackeling, Gavin: *Mastering Machine Learning with scikit-learn.* Packt Publishing Ltd, 2014

[52] Hossinger, Reinhard ; Heimbuchner, Klaus ; Uhlmann, Tina: Development of a Real-time Model of the Utilisation of Short-term Parking Zones. In: *19th ITS World Congress*, 2012

[53] Ji, Yanjie ; Tang, Dounan ; Blythe, Phil ; Guo, Weihong ; Wang, Wei: Short-term forecasting of available parking space using wavelet neural network model. In: *IET Intelligent Transport Systems* 9 (2014), Nr. 2, S. 202–209

[54] Klappenecker, Andreas ; Lee, Hyunyoung ; Welch, Jennifer L.: Finding available parking spaces made easy. In: *Ad Hoc Networks* 12 (2014), S. 243–249

[55] Koster, Andrew ; Oliveira, Allysson ; Volpato, Orlando ; Delvequio, Viviane ; Koch, Fernando: Recognition and recommendation of parking places. In: *Ibero-American Conference on Artificial Intelligence* Springer, 2014, S. 675–685

[56] Kotb, Amir O. ; Shen, Yao-Chun ; Zhu, Xu ; Huang, Yi: iParker—A New Smart Car-Parking System Based on Dynamic Resource Allocation and Pricing. In: *IEEE Transactions on Intelligent Transportation Systems* 17 (2016), Nr. 9, S. 2637–2647

[57] Lin, Trista ; Rivano, Hervé ; Le Mouël, Frédéric: A Survey of Smart Parking Solutions. In: *IEEE Transactions on Intelligent Transportation Systems* (2017)

[58] Lin, Trista S.: *Smart parking: Network, infrastructure and urban service*, Lyon, INSA, Diss., 2015

[59] Mathur, Suhas ; Jin, Tong ; Kasturirangan, Nikhil ; Chandrasekaran, Janani ; Xue, Wenzhi ; Gruteser, Marco ; Trappe, Wade: Parknet: drive-by sensing of road-side parking statistics. In: *Proceedings of the 8th international conference on Mobile systems, applications, and services* ACM, 2010, S. 123–136

[60] Nandugudi, Anandatirtha ; Ki, Taeyeon ; Nuessle, Carl ; Challen, Geoffrey: Pocketparker: Pocketsourcing parking lot availability. In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing* ACM, 2014, S. 963–973

[61] Pullola, Sherisha ; Atrey, Pradeep K. ; El Saddik, Abdulmotaleb: Towards an intelligent GPS-based vehicle navigation system for finding street parking lots. In: *Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on* IEEE, 2007, S. 1251–1254

[62] Rajabioun, Tooraj ; Foster, Brandon ; Ioannou, Petros: Intelligent parking assist. In: *Control & Automation (MED), 2013 21st Mediterranean Conference on* IEEE, 2013, S. 1156–1161

[63] Rajabioun, Tooraj ; Ioannou, Petros A.: On-street and off-street parking availability prediction using multivariate spatiotemporal models. In: *IEEE Transactions on Intelligent Transportation Systems* 16 (2015), Nr. 5, S. 2913–2924

[64] Richter, Felix ; Di Martino, Sergio ; Mattfeld, Dirk C.: Temporal and spatial clustering for a parking prediction service. In: *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on* IEEE, 2014, S. 278–282

[65] Shin, Jong-Ho ; Jun, Hong-Bae: A study on smart parking guidance algorithm. In: *Transportation Research Part C: Emerging Technologies* 44 (2014), S. 299–317

[66] Szczurek, Piotr ; Xu, Bo ; Wolfson, Ouri ; Lin, Jie ; Rishe, Naphtali: Learning the relevance of parking information in VANETs. In: *Proceedings of the seventh ACM international workshop on VehiculAr InterNETworking* ACM, 2010, S. 81–82

[67] Tiedemann, Tim ; Vögele, Thomas ; Krell, Mario M. ; Metzen, Jan H. ; Kirchner, Frank: Concept of a Data Thread Based Parking Space Occupancy

Prediction in a Berlin Pilot Region. In: *AAAI Workshop: AI for Transportation*, 2015

[68] Vlahogianni, Eleni I. ; Kepaptsoglou, Kostanstinos ; Tsetsos, Vassileios ; Karlaftis, Matthew G.: Exploiting new sensor technologies for real-time parking prediction in urban areas. In: *Transportation Research Board 93rd Annual Meeting Compendium of Papers*, 2014, S. 14–1673

[69] Witten, Ian H. ; Frank, Eibe ; Hall, Mark A. ; Pal, Christopher J.: *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2016

[70] Wu, Eric Hsiao-Kuang ; Sahoo, Jagruti ; Liu, Chi-Yun ; Jin, Ming-Hui ; Lin, Shu-Hui: Agile urban parking recommendation service for intelligent vehicular guiding system. In: *IEEE Intelligent Transportation Systems Magazine* 6 (2014), Nr. 1, S. 35–49

[71] Xu, Bo ; Wolfson, Ouri ; Yang, Jie ; Stenneth, Leon ; Philip, S Y. ; Nelson, Peter C.: Real-time street parking availability estimation. In: *Mobile Data Management (MDM), 2013 IEEE 14th International Conference on* Bd. 1 IEEE, 2013, S. 16–25

[72] Zheng, Yanxu ; Rajasegarar, Sutharshan ; Leckie, Christopher: Parking availability prediction for sensor-enabled car parks in smart cities. In: *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 IEEE Tenth International Conference on* IEEE, 2015, S. 1–6