Capstone Project

Machine Learning Engineer Nanodegree

Christian Wolff

1 Definition

1.1 Project Overview

In recent years, bike-share programs in cities around the world have grown in popularity. Wikipedia[1] currently lists 330 bike-sharing systems across the world. Their ultimate goal is to create a more sustainable transportation landscape that provides new mobility options for short trips and to improve connectivity to other modes of transportation.

As bike-sharing programs grow in popularity, cities begin to realize additional benefits which should mean plenty of related opportunities for years to come. But there are also challenges to overcome. One of the challenges are the running costs of said programmes, mostly from maintenance of bikes and stations, redistribution of bikes by vans during the day and the necessary IT infrastructure.

A Hubway station

In this project, we will take a closer look at Hubway[2], Boston's regional bike sharing system. As of September 2017, it gives members access to more than 1,700 bikes at 170 stations. The station network is designed to offer a one-way transportation option and an alternative to public transport like bus or train.

1.2 Problem Statement

Bike-sharing services can only be successful when they meet the demand for mobility. In this regard, two important factors for customer satisfaction are

- high availability of bikes at all stations,
- and the possibility to return a bike to any station.

But cities are one of the most complex systems known to us, and far from being homogeneous or in equilibrium. Thus, some bike-sharing stations are frequented more often then others, with changing distributions over the day and week. This means, that bikes have to be continuously redistributed to meet the demand.

Hubway tackles this problem on a real-time basis, as we can read on their website:

> Hubway employs 4-5 rebalancing vans, each with a payload of 20-25 bikes, used to redistribute bicycles between 6am and 10pm, 7 days a week. Bicycles are redistributed dynamically in response to real-time data.

[1] https://en.wikipedia.org/wiki/List_of_bicycle-sharing_systems#Cities
[2] https://www.thehubway.com

From this statement follows the question:

Can Hubway predict the demand for bikes and plan the necessary rebalancing ahead of time?

A Hubway rebalancing van

We can assume that proper planning of rebalancing will decrease running costs and improve customer satisfaction. The goal of this project is the implementation of a workflow to predict the daily demand of bikes at the stations, so that rebalancing can be planned ahead of time, instead of relying on real-time rebalancing.

The first part of the following project documentation explains the datasets and how they are transformed for machine learning. We will see that a visual analysis of the data can help in its preparation for the learning task by removing data of low quality and time-dependent environmental effects. The first part concludes with an explanation how the overall demand for bikes can be derived from individual bike trips.

In the second part, the rebalancing problem is stated as a regression task, with the number of bike departures and arrivals at each station as target variables. First, single decision trees without further refinement are trained as benchmark. Then the boosted tree algorithm from the popular XGBoost library is used for the final model training and a method for parameter refinement via grid search is explained.

The third part shows how the trained model can be integrated into an interactive web application that helps to solve the rebalancing problem in a real-world scenario. This application gives the user an overview of bike movements for a selected date, and the option to drill down into a single station to reach a detailed view. The overall design of the application shall give insight into historical data and also shows real-time predictions of future events.

The fourth and last part of this documentation comments on the results of the previous parts. Difficulties are discussed and suggestions for further improvement are given.

1.3 Metrics

The data used for model learning is a time series of bike trips. For the training and evaluation of the predictive model, the dataset will be split at a certain date into two sets. The set with the older data will be used for training the model, the younger set for testing. This ensures that the model can predict future developments from the past.

The target value of the model will be the demand for bikes at a station for given day and hour, which is a regression problem. The root of the mean squared error over all stations and dates of the test set will be used to quantify the success of the model. The RMSE is defined as

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

where N is the number of samples, y_i is the observed value for the ith observation and $\hat{y}_i$ is the prediction. Because the difference of real value and prediction is squared for each sample, the RMSE penalizes larger prediction errors more. This property is desirable for the ensemble method used in this project, because it will help to quickly focus the training process in erroneous predictions and speed up learning.

2 Analysis

2.1 Data Exploration

2.1.1 Bike Trips Dataset

From the very beginning of their bike-sharing program, Hubway has collected usage data. In 2012, they ran a public challenge to visualize and analyze the data. The results of the challenge are available on Hubway's official challenge website[3].

Hubway publishes trip data every quarter on its official system data page[4]. The main data source for this project is data from January 2015 through November 2016, containing over 2.3 million bike trips collected at almost 200 stations. The set contains fine-grained data of single trips, e.g. start and end timestamp of each trip, together with the respective station.

The total number of departures and arrivals for each station and hour of day will be extracted from this dataset. The fields `tripduration`, `bikeid` and all customer information will not be used and can be dropped. This procedure and the statistics of the resulting flow dataset are described under section 3.1.1 Flow Extraction.

Field definitions in Hubway trip data	
`tripduration`	Time of trip in seconds.
`starttime`	Start timestamp of trip with date and time, in EST.
`stoptime`	End timestamp of trip with date and time, in EST.
`start station id`	ID of the departure station.
`start station name`	Descriptive name of the departure station.
`start station latitude`	Latitude of the departure station.
`start station longitude`	Longitude of the departure station.
`end station id`	Station ID of the arrival station.
`end station name`	Descriptive name of the arrival station.
`end station latitude`	Latitude of the arrival station.
`end station longitude`	Longitude of the arrival station.
`bikeid`	ID of bicycle used.
`usertype`	Subscription type. "Registered" is a user with membership. "Casual" is a user without membership.
`birth year`	Birth year of the user, self-reported by member
`gender`	Gender of the user, self-reported by member

2.1.2 Bike Stations

For the web application all stations need to be shown on an interactive map. This requires a compiled dataset of stations with descriptions and coordinates. Hubway provides this information as a file (`/data/raw/Hubway_Stations_2011_2016`). Upon closer inspection it turned out that the station IDs in this file are different from the IDs used in the trip dataset. For example, the station 'Fan Pier' has the ID A32000 in the station dataset. In the trips dataset the ID 7 is used instead.

Because of this inconsistency, a decision was made to extract the station list directly from the trips dataset. The necessary steps can be found in the Python notebook `03_stationExtraction`. A cross-check with Hubway's provided list of stations showed that the station with ID 1 is Hubway's central storage warehouse. The trips dataset contains 486 trips with the storage warehouse as arrival station, marking trips that where either done for repair or storage of a bike. Because of the comparably small number of these maintenance trips, they where removed from the dataset.

In addition to station ID and description, the dates of first and last occurrence of each station where extracted. These dates indicate the time period in which a station was operational. From this data it became

[3] http://hubwaydatachallenge.org
[4] https://www.thehubway.com/system-data

clear that the number of stations varies over time, which will be discussed further under 2.2 Exploratory Visualization.

To distinguish stations we can use either the station ID, which is a categorical feature that was assigned arbitrarily by Hubway and has no specific order structure corresponding to real-world information. Or we use the station coordinates, which impose a topological order onto the dataset. Because this natural ordering can help machine learning algorithms to extract meaningful information, we will use PCA-transformed coordinates during the learning phase.

Field definitions of station data	
`station_id`	Station ID from the trips dataset.
`station_name`	Station description.
`latitude`	Latitude of the station.
`longitude`	Longitude of the station.
`first_used`	First occurence of the station ID in the trips dataset.
`last_used`	Last occurence of the station ID in the trips dataset.

2.1.3 Additional Feature Sets

Since cycling is an outdoor activity, the bike trips are very likely correlated with the local weather in Boston. Hourly historic weather summaries where acquired from DarkSky[5].

Field definitions of weather summaries	
`timestamp`	Date and time of the full hour.
`apparentTemperature`	The apparent (or 'feels like') temperature in degrees Celcius.
`cloudCover`	The percentage of sky occluded by clouds, between 0 and 1, inclusive.
`dewPoint`	The dew point in degrees Celcius.
`humidity`	The relative humidity, between 0 and 1, inclusive.
`icon`	A machine-readable text summary of this data point, suitable for selecting an icon for display. If defined, this property will have one of the following values: clear-day, clear-night, rain, snow, sleet, wind, fog, cloudy, partly-cloudy-day, or partly-cloudy-night.
`precipAccumulation`	The amount of snowfall accumulation expected to occur, in inches. (If no snowfall is expected, this property will not be defined.)
`precipIntensity`	The intensity (in inches of liquid water per hour) of precipitation occurring at the given time. This value is conditional on probability (that is, assuming any precipitation occurs at all) for minutely data points, and unconditional otherwise.
`precipProbability`	The probability of precipitation occurring, between 0 and 1, inclusive.
`precipType`	The type of precipitation occurring at the given time. If defined, this property will have one of the following values: 'rain', 'snow', or 'sleet' (which refers to each of freezing rain, ice pellets, and 'wintery mix'). (If precipIntensity is zero, then this property will not be defined. Additionally, due to the lack of data in our sources, historical precipType information is usually estimated, rather than observed.)
`pressure`	The sea-level air pressure in millibars.
`summary`	A human-readable text summary of this data point.
`temperature`	The air temperature in degrees Celcius.
`uvIndex`	The UV index.
`visibility`	The average visibility in kilometers, capped at 10 kilometers.
`windBearing`	The direction that the wind is coming from in degrees, with true north at 0° and progressing clockwise. (If windSpeed is zero, then this value will not be defined.)
`windSpeed`	The wind speed in kilometers per hour.

[5] https://darksky.net

The second dataset used for augmentation are public holidays for 2015 and 2016. We can expect a baseline of regular commuters during work days, and spikes of casual usage during weekends and public holidays. Information about public holidays in Boston can be acquired from the Massachusetts state government[6].

Field definitions of public holiday data	
`description`	Common descriptive name of the public holiday.
`date`	Date of the public holiday.

2.2 Exploratory Visualization

2.2.1 Bike Stations

The number of stations in operation per month can be aggregated from the extracted station list. The result of this aggregation is shown in figure 1, with data points for the 1st and 15th of each month. Until April 2015, close to 60 stations where operated by Hubway. In May, this number was more than doubled, with a steady increase over the following months.

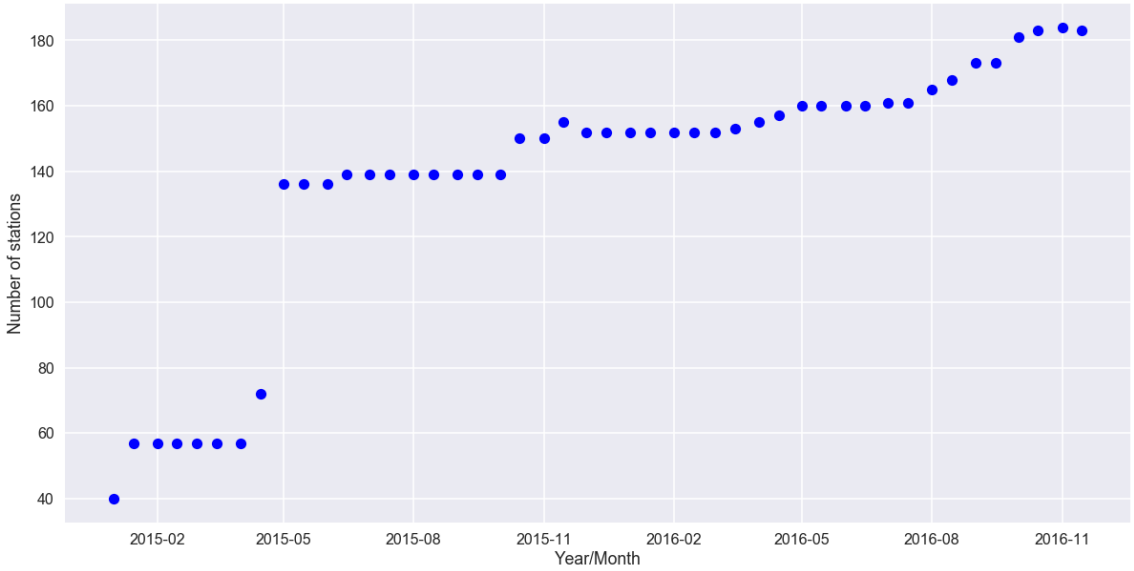


Figure 1: Number of stations per year and month

This observation leads to two conclusions:

- The sudden increase of station numbers in May 2015 is a major change of the whole data environment. It can be argued that a predictive model should not be trained with data that was gathered in a fundamentally different environment. Therefore, only data from May 2015 through November 2016 is used for model training.

- The change in numbers from May 2015 onward raises the following question: How does a newly opened station affect the usage of already established stations that are close by? Depending on the distance to surrounding stations, the new station will take over some of their load. In other words, when looking at a specific area, we expect the overall demand to be constant and equally shared by all stations in that area. An approach to handle these changes with pooled clusters is discussed under section 5.2.1 Clustering. For the sake of simplicity, we will use only those stations that where operational during the whole period from May 2015 through November 2016, and remove all other stations from the dataset.

[6]https://www.mass.gov/

2.2.2 Bike Trips

Figure 2 shows the total number of departing trips for each month, separated by year. In this plot we can see that significantly more trips are taken during the summer months, which is consistent with our expectations.

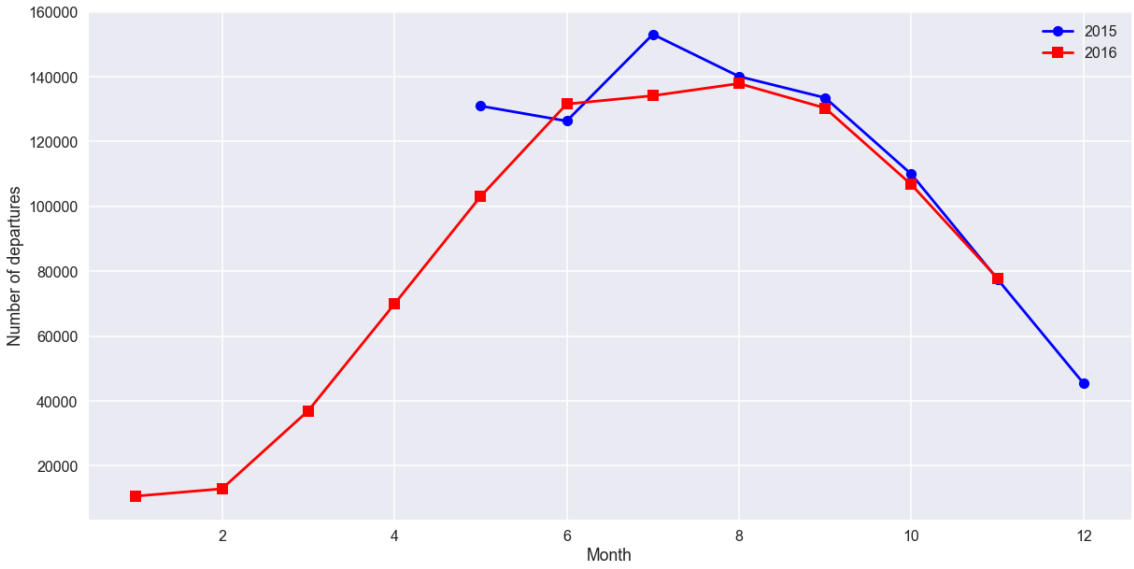


Figure 2: Total number of trips per month

The full dataset contains two summer periods, but only one winter period. To avoid this bias, all further visualisations will be taken from a reduced dataset, from May 2015 to June 2016. The boxplot in figure 3 shows the number of departing trips taken per hour. The box statistic for each hour category is calculated over the 12 months. The whiskers extend to the extreme values of the distribution and the box shows the interquartiles at 25%, median and 75%. From midnight until the early morning hours we find almost no activity, and most trips are taking around the start and end of workdays, 8 am and 5 pm respectively.

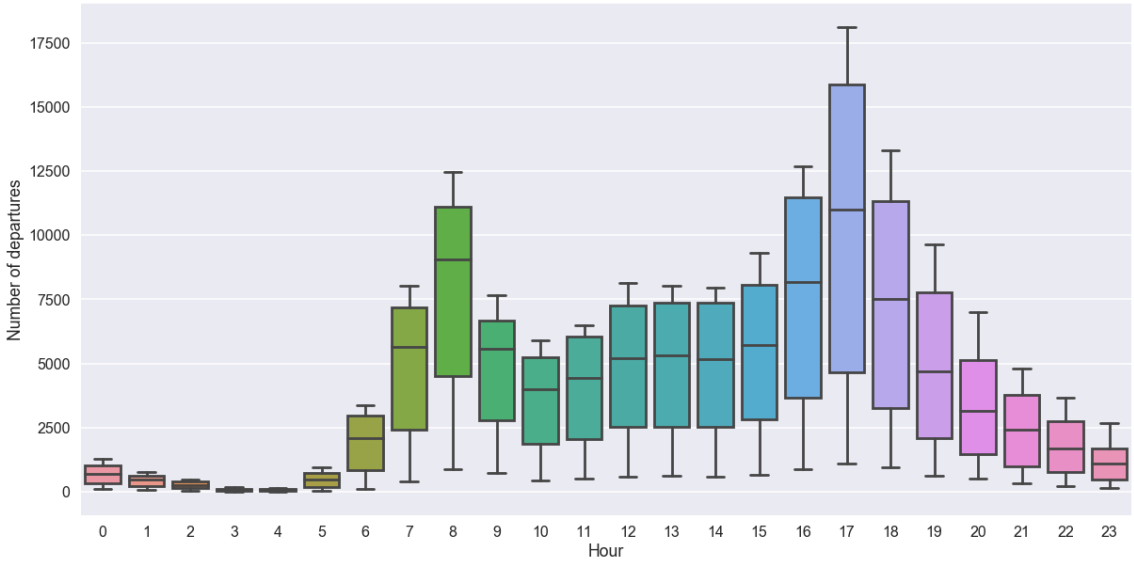
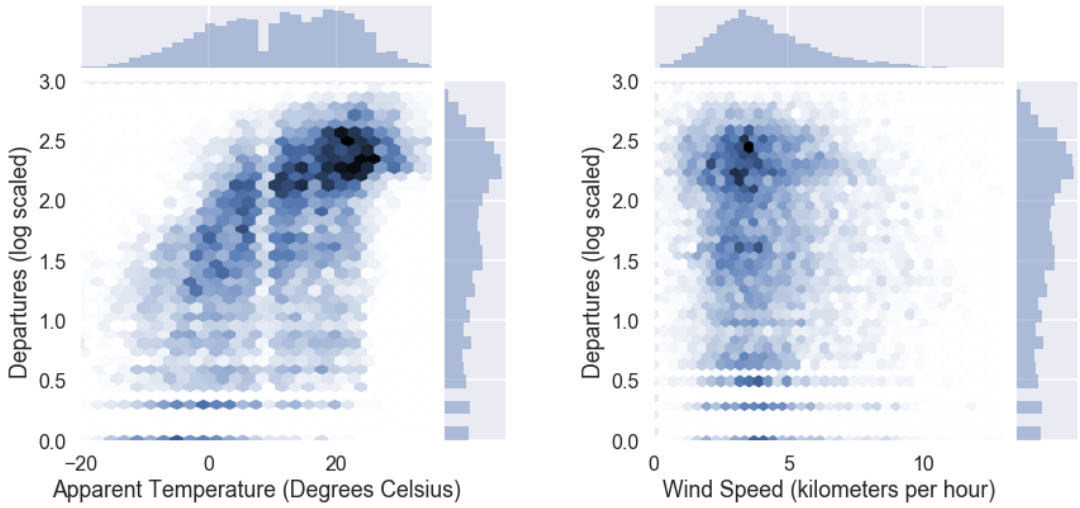


Figure 3: Total number of trips per hour of the day

2.2.3 Local Weather

Figure 4 shows the correlation of weather effects with the total number of departures. The number of departures where log-scaled, due to their wide numerical range. We see that most trips are taken when tem-

peratures are mild (around 20° Celsius) and customers tend avoid extreme temperatures. The distribution for wind speed can be interpreted as a Poisson distribution with its peak around 3 kilometers per hour, which can be explained with Boston's location at the sea and relatively flat geography. We can expect that wind speed does not provide as much information as temperature, and that temperature has more importance for model learning.



(a) Apparent Temperature

(b) Wind Speed

Figure 4: Hexbin plots of local weather and total number of departures

2.3 Algorithms and Techniques

2.3.1 Principal Component Analysis

Principal component analysis (PCA) can be defined as an orthogonal projection of data onto a space of same or lower dimension, such that the variance of the projected data is maximized along the main axes. PCA is used for dimensionality reduction, feature extraction, data compression and visualization. We use it to get a transformed set of coordinates from the original station coordinates (latitude and longitude), with the transformed coordinates having maximum variance along the new main axes. The transformation keeps the topological order of stations intact and gives an advantage to algorithms that try to partition the data (like decision trees).

2.3.2 Model

For the actual model learning we use XGBoost, a boosted tree model. Boosting techniques can mitigate the negative properties of a single classifier by creating ensembles of many weak classifiers, in this case small decision trees that do not overfit. Predictions are then made by combining the results of all predictors, much like the taking the vote of a committee. During the iterative training stage, Boosting gives more weight to misclassified training instances to allow new classifiers to focus on the harder cases. This allows each classifier to build expertise on a small part of the training data, which increases the overall models performance when all simple classifiers are combined. Tree Boosting is a good candidate for the problem, because the dataset is large, but clean. This property of the data minimizes the chance that some of the weak learners will focus on outliers and thereby maximizing the overall accuracy on the testing data.

As objective function we use Poisson regression, which is best suited for count data. This means that only positive numbers can be used as regression targets and we need to learn two separate models for arrivals and departures. The models will output the mean of the learned Poisson distributions as predictions, and we can compute the predicted flow from their difference.

3 Methodology

3.1 Data Preprocessing

3.1.1 Flow Extraction

Before transforming the dataset of bike trips into flow data, the following adjustments are made:

- Removal of all features that can not be aggregated by hour. These are the fields `tripduration`, `bikeid` and all customer information.
- The station with ID 1 is Hubway's storage warehouse and all trips to or from this location are removed.
- Due to the significant rise in the number of available stations in May 2015, only trips from May 2015 through November 2016 are used (cf. section 2.2.1 Bike Stations).
- Only data from stations that where active throughout the above mentioned time period is used, thus reducing the number of stations from 189 to 119.
- 9 trips that ended on the 1st of December 2016, shortly after midnight, where removed.
- 3 trips with locations outside of Boston where removed. These where 'dummy' locations used by Hubway's technicians for testing.

After these necessary adjustments, the number of arrivals and departures of bikes at each station per hour is calculated by aggregation of the trips data. The flow at a station during a given hour is defined as the number of arrivals minus the number of departures. All steps can be found in the Python notebook `05_flowExtraction`.

Trip ID	Start time	Start station	End time	End station
1	7/28/2011 12:11:00	2	7/28/2011 12:23:00	8
2	7/28/2011 16:08:00	5	7/28/2011 16:20:00	4
3	7/28/2011 19:18:00	8	7/28/2011 19:46:00	1

From trips

to flows

	Arrivals	Departures	Flow
Midnight to 6am	0	1	-1
6am to 12am	0	3	-3
12am to 6pm	5	3	2
6pm to midnight	6	2	4

(No causal relation between the tables.
Flow data is for one day and one station.
Hourly data condensed into four time periods for illustration purposes.)

Figure 5: Transformation of trips into flows

3.1.2 Additional Feature Sets

The feature sets containing weather and public holiday data are pre-processed as follows:

- Date features are created from the field `timestamp`: `date`, `hour`, `weekday`, `month`, `year`. These features can be used for convenient filtering, creating new features or directly in the learning process.
- From the public holiday dataset the following boolean features are created: `is_holiday`, `is_weekend`, `is_weekend_or_holiday`. These are used as features in the learning process.
- The features `cloudCover` and `uvIndex` contain unknown values and are removed, because the benchmark model is unable to process these.

3.2 Implementation

All steps from data acquisition until the training of the final models and their refinement where implemented in Jupyter notebooks on Python 3. The organization of the files is as follows:

- `00_darkskyWeatherAquisition`

  Acquisition of weather data from the Darksky API (the API key was removed for the submission). The weather data is stored in json format. Particular care must be taken when handling timestamps in the subsequent data transformations steps, because timezones have to be set for correct merging of datasets.

- `01_tripDataPreparation`

  Concatenation of trip files into a single dataset and removal of unused features. Almost 500 trips, that where made by Hubway's technicians and not officially documented in their data description, where also removed.

- `02_additionalDataPreparation`

  Creates full additional feature set from weather and holiday data.

- `03_stationExtraction`

  Extracts station names and locations from the trips dataset. This is necessary because the station IDs provided in Hubway's data files are not consistent. Also determines for each station its active time period and saves the result to a csv file. Because stations are matched by their full descriptive text, care must be taken that the results are correct, e.g. by comparing geographic locations.

- `04_stationVizAndTripAdjustment`

  Plots the total number of active stations per month. Adjustments of the trip dataset to account for the changing number of stations.

- `05_flowExtraction`

  Computes arrivals and departures from the reduced trips dataset. Fills missing date-hour combinations with empty dummy entries. Again, setting the correct timezone is important for data merge operations.

- `06_visualisation`

  Plots for section 2.2 Exploratory Visualization. The visual exploration led to the insight that the overall number of stations changed over time and a huge portion of the data was removed as a result of this.

- `07_PCA`

  Transformation of station coordinates according to their principal component analysis. Due to the well documented machine learning libraries, this part and the following where easy and straightforward to implement.

- `08_benchmarkModel`

  Training and evaluation of the single tree benchmark model, as described under section 3.3 Benchmark.

- `09_modelTrainingDepartures` and `10_modelTrainingArrivals`

  Training and evaluation of the XGBoost models for arrivals and departures. Also contains the code used for parameter refinement. Saves both models to the file system.

3.3 Benchmark

To create an initial benchmark, a single decision tree was trained after pre-processing the data as described under section 3.1 Data Preprocessing. The standard DecisionTreeRegressor from scikit-learn was used and no refinement was done. Two separate models for the number of arrivals and departures where trained. The training set consisted of the data from May 2015 through April 2016, and the trained models where tested against the data from May 2016 through November 2016. Results for the test set are shown in table 1 Results for the benchmark models.

	RMSE	R^2
Arrivals model	2.163	0.256
Departures model	2.193	0.182

Table 1: Results for the benchmark models

3.4 Refinement

XGBoost provides wrappers for scikit-learn, making it easy to use with the standard parameter optimization methods. We used grid search with cross-validation to find an optimal parameter combination.

A reduced training dataset of 10.000 entries was created to speed up the search. Furthermore, only data from May 2015 through April 2016 is used for the cross-validated grid search. The rest of the data serves as test set for the initial and the final model.

Parameter optimization followed these steps:

1. A model with initial parameters is crated and trained on the full dataset. This model serves as a benchmark for the final model to check if the parameter optimization is valid.
2. A grid search on the reduced dataset is conducted, with given parameter combinations and high learning rate. Usually one or two parameters are optimized at a time.
3. Step 2 is repeated until all parameters are optimized.
4. The final model is trained and its performance checked against the initial model from step 1.

The following parameters where optimized:

1. `max_depth`

   The maximum depth of a tree. A higher depth will allow the model to learn relations very specific to a particular sample.

   Initial value: 3

2. `min_child_weight`

   Sets the minimum sum of weights of all observations required in a child. Higher values prevent a model from learning relations which might be highly specific to a particular sample.

   Initial value: 1

3. `gamma`

   Gamma specifies the minimum loss reduction required to make a node split.

   Initial value: 0

4. `subsample`

   Sets the fraction of observations to be randomly sampled from the training set for a tree.

   Initial value: 0.8

5. `colsample_bytree`

   Sets the fraction of columns to be randomly sampled for a tree.

   Initial value: 0.8

6. `reg_alpha`

   L1 regularization term on weights.

   Initial value: 0

Table 2 shows the progession of parameter tuning and the final parameter values. A direct comparison of the RMSE scores with the scores of the benchmark or final model is not possible, because the tuning scores where taken on the training set, which was further divided into folds by the grid search algorithm.

	Arrivals		Departures	
Tuned Parameter	Final Value	RMSE	Final Value	RMSE
`max_depth` `min_child_weight`	6 7	1.336	5 4	1.364
`gamma`	0.2	1.334	0.7	1.362_{087}
`subsample` `colsample_bytree`	1 1	1.326_{420}	0.8 0.9	1.362_{072}
`reg_alpha`	0.1	1.325_{639}	10^{-4}	1.361_{696}

Table 2: Parameter Tuning

4 Results

4.1 Model Evaluation and Validation

All models where trained on the full dataset, containing $1.045.415$ samples in the training set and 611.184 samples in the test set. Table 3 Model Comparison shows the scores on the test set for the benchmark, initial model and the final model. Both RMSE and R^2 score are given, with R^2 being a coefficient that measures how much variance in the target variable can be predicted by the model, with a possible maximum value of 1.

Objective	Model	RMSE	R^2
Arrivals Counter	Single Decision Tree (Benchmark)	2.163	0.256
	Initial XGBoost Model	1.626	0.580
	Final XGBoost Model	1.605	0.591
Departures Counter	Single Decision Tree (Benchmark)	2.193	0.182
	Initial XGBoost Model	1.628	0.549
	Final XGBoost Model	1.584	0.573

Table 3: Model Comparison

Figure 6 Feature Importance for the Arrivals Model shows all used features, sorted by their relevance for the model for the arrivals counter. The transformed station location is the most important feature, followed by the hour of the day. Two weather features come next, namely apparent temperature and pressure. Information about public holidays is not as important as expected, which is probably due to the fact that there are not many public holidays in comparison to regular workdays and weekends.

Because boosting methods use ensembles of many weak, but specialized learners, they tend to avoid overfitting. The final models also show this property, which is expressed by the comparison of the RMSE of the training data to the test data. The RMSE after training the arrivals model was 1.283, and for departures 1.243, where both measurements where taken on the training data. We see that these scores are lower than the RMSE of the test set, which can be expected, but the difference is only around 27%. This and the fact that the weak learners are decision trees, which can handle outliers well, makes the models robust with regard to unseen data. Furthermore, the data has at least two periodical trends, annually and daily, which help the model to find patterns and make it more robust.

4.2 Web Application

A web application was designed to complete the whole end-to-end scenario from data acquisition to model learning to analysis by an end-user. The application gives more insight into the data and the predictions by showing all stations on a map and displaying the real historical data and predictions in tabular form. By default, an overview of all stations is displayed, which shows the accumulated data of all stations. Users can drill-down to detailed data of a station by clicking on its map marker.

The web application comes with a server, which was written in Python using the flask library. The server is responsible for making predictions using the trained models. It also sends the historical data to the web application and serves all static files. The full web application can be found in the directory `webapp` of this project's GitHub repository[7]. Instructions for running the server are included in the `README` file. The application itself has a help function that guides users through its main features.

To show how the application can help the rebalancing task in a real-world scenario, it indicates the hours with critical events. These events are defined as station flows with an absolute value greater than some threshold. In the current implementation the threshold is set to 6. On the overview the markers for critical events can be clicked to highlight all stations on the map with such events. This information can then be used to drill down further into the station's details.

[7] https://github.com/levkazar/udacityMLCapstone

4.3 Justification

We see clear improvements from the benchmark model through parameter optimization to the final model. The RMSE of the final models is approximately 1.6. When making predictions and rounding to the nearest number, we can expect an error of ± 2 for both arrivals and departures. When calculating the flow from the difference of arrivals and departures, the maximum expected error is 4. This result shows that its possible to learn the patterns in the given data, but it is not good enough yet to serve as a tool for real-world decision making.

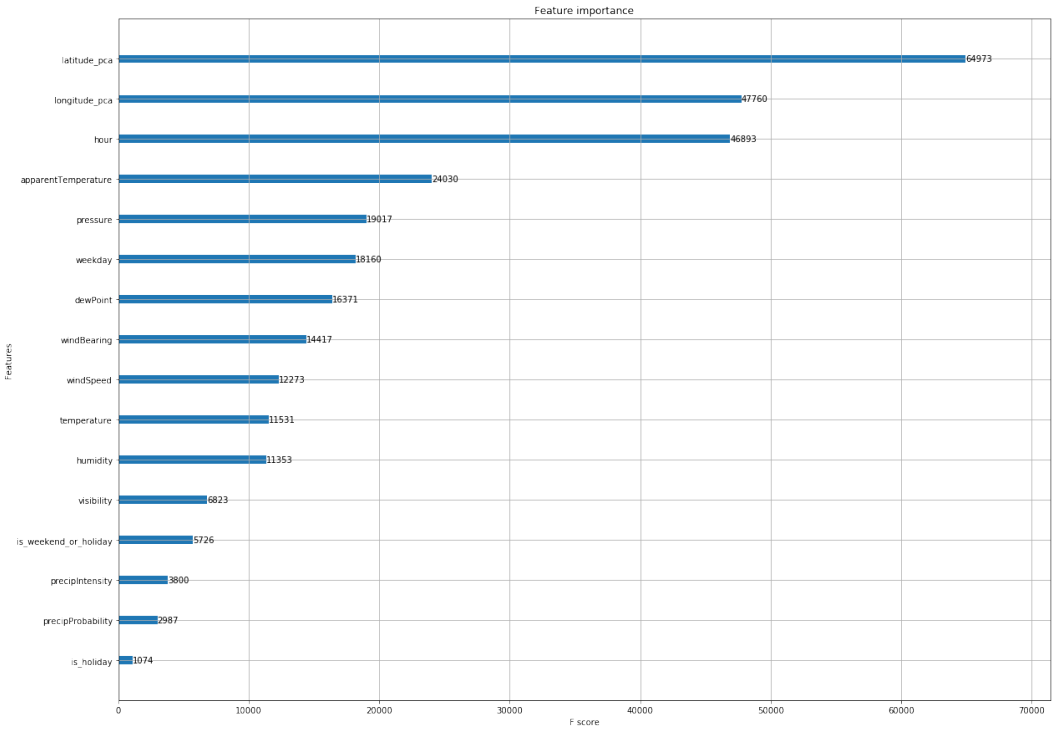


Figure 6: Feature Importance for the Arrivals Model

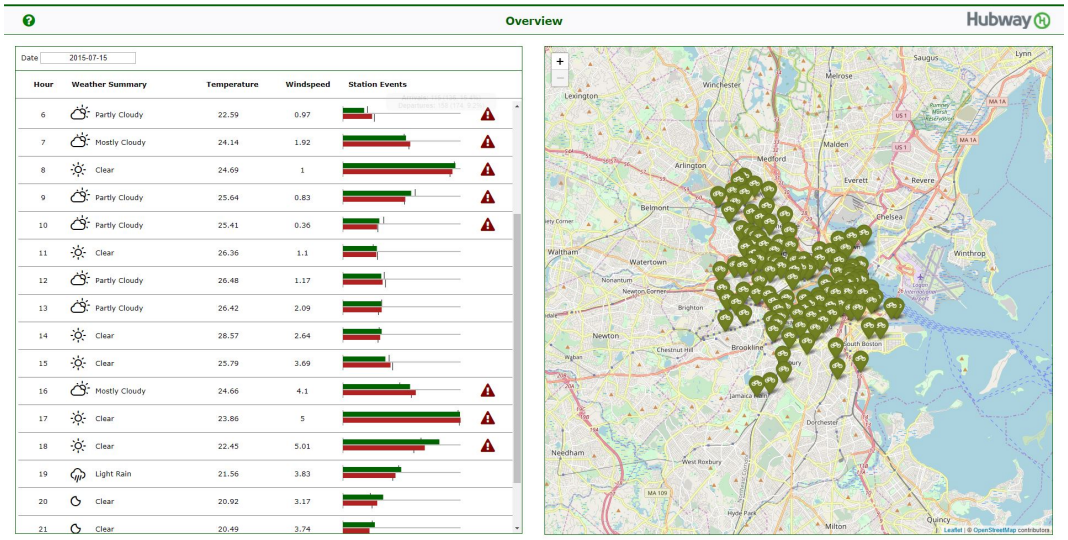


Figure 7: Web application

5 Conclusion

5.1 Reflection

This project presented an end-to-end solution for the rebalancing problem in bike-sharing. We started with the trip data provided by Hubway and ended up with a web application to visualize predictions of the trained models. Each step of this project had its own difficulties and challenges:

1. **Data acquisition.** Finding a provider of historical, hourly weather data with an API for mass data proved to be difficult. Most providers can give only daily summaries of historical weather, or they restrict access to downloadable files of short time periods.

2. **Data cleansing.** Hubway states that its public system data is clean and ready-to-use. After comparing the files for trip and station data, it became clear that some of the trips where done for maintenance and that station IDs did not match. The first attempt of finding a common set of station IDs then show that the number of active stations varied over time. After the necessary adjustments, we arrived at reduced dataset taken from two-thirds of all stations and cutting off the first 4 months of the original data.

3. **Model training.** Due to the well documented APIs of the machine learning libraries the model training proceeded without problems. Initially, only the station ID was used as a feature to discriminate stations. The resulting XGBoost models did not show a significant improvement over the benchmark models. After adding geo location to the feature list, which provides a natural sorting, model results improved.

4. **Building the web application.** Machine learning can be used as a supportive tool for analysis and planning. The web application was implemented as the concluding step to the whole end-to-end solution and shows how the models' predictions can be presented to an end-user. This step of the project provided a challenge, because it required not only different technologies, but also a different mindset with regard to program logic and design.

Overall, the greatest challenge lay in the integration of all parts into a coherent whole. Even though the general direction and goals where clear from the beginning, the project soon entered an iterative loop, with refinements being done in the earlier process steps, because of discoveries in the later steps. Most changes had to be done in the preparation of the data, most probably because it is more an art than a solid science and can only be learned by experience. And because data preparation is such an important step towards a well trained model, gaining more experience with statistical methods and data visualization techniques from real-world problems is a key skill for anyone working in the field of machine learning.

5.2 Improvement

5.2.1 Clustering

We saw in 2.2.1 that the number of stations changed over the 2 year period. As a result, we kept only those stations for the learning tasks that where active during the whole period. This reduced their number by more than a third, from 189 to 119 and made a lot of data unavailable for training. A better solution would be a cluster model with a reduced number of surrogate stations represented by the cluster centers. The clusters can be created with a Gaussian Mixture model from the full set of coordinates of all stations, thus arriving at a time-independent representation. The predicted posterior probabilities for all real stations would then be used for a weighted pooling of the data. The training of the XGBoost models would then be done one the pooled data of the surrogate stations. When making predictions, we would have to map from the pooled model back to the real station data, using the weights from Gaussian Mixture model.

The clustered approach would give us access to all historical data and would probably result in a better predictive model. But we would also have to deal with two separate models, the cluster and the predictive model, which need to be optimized separately or in an iterative loop. Furthermore, mapping the predicted data back to the real stations is important for the evaluation of the whole model. Though it is a standard problem from linear algebra, this step can significantly increase processing times.