Capstone Project
Machine Learning Engineer Nanodegree

Demand prediction for a bike sharing systems

Mai 2016
Philipp Vogler

1. Definition

1.1 Project Overview

- A Problem in the area of transport and logistics that is solvable with machine learning.
- Utilizing machine learning to forecast the demand for the Washington DC bike sharing system 'capital bike share'.
- Using different types of regression to find an algorithm to predict the demand for bikes based on calendric and weather information.
- Weather, calendar and demand information is provided in a dataset by the University of Porto at UCI ML Repository.
- This project tries to create a forecasting function based on two years of historical data by utilizing the machine learning libraries scikit-learn and tensor-flow.

> http://www.capitalbikeshare.com
> http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset
> http://freemeteo.de/wetter/
> http://dchr.dc.gov/page/holiday-schedules
> http://scikit-learn.org/stable/
> https://www.tensorflow.org

1.2 Problem Statement

The goal is to forecast the demand for bikes in dependency of weather conditions like outside temperature and calendric informations e.g. holidays. These information and the demand structure is provided in a set with two years of daily historic data.
The demand is given as the total daily demand and as a split for registered users and casual users. To increase the quality of the prediction registered user demand and casual user demand will be predicted separately in step two.
To make predictions machine learning is used to train regressors. Scikit-Learn recommends a support vector regressor (SVR) for this kind of problem and data amount. In addition a deep neuronal network (DNN)

regressor is trained for comparison. To find the hyper-parameters for these regressors grid search and randomized search are utilized. Due to the small dataset cross validation is applied.

> http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
> http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR
> https://github.com/tensorflow/skflow/blob/master/g3doc/api_docs/python/estimators.md
> http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html
> http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.RandomizedSearchCV.html

1.3 Metrics

To measure the performance of the regressions two standard regression metrics are used: Mean squared eror (MSE) and the coefficient of determination (R^2). Both metrics are calculated for both regressor types. For comparison and parameter tuning only R^2 is used due to the better readability.

> http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html#sklearn.metrics.mean_squared_error
> http://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html#sklearn.metrics.r2_score

2 Analysis

2.1 Data Exploration

```
Feature column(s):
'instant'
‘dteday'
'season’
‘yr'
'mnth'
'holiday'
'weekday'
'workingday'
'weathersit'
'temp'
'atemp'
'hum'
'windspeed'
```

Target column:
```
cnt
```

Data values:
```
   instant      dteday  season  yr  mnth  holiday
weekday  workingday  \
0        1  2011-01-01       1   0     1        0
6           0
1        2  2011-01-02       1   0     1        0
0           0
2        3  2011-01-03       1   0     1        0
1           1
3        4  2011-01-04       1   0     1        0
2           1
4        5  2011-01-05       1   0     1        0
3           1

   weathersit      temp     atemp       hum
windspeed  casual  registered  \
0           2  0.344167  0.363625  0.805833
0.160446     331         654
1           2  0.363478  0.353739  0.696087
0.248539     131         670
2           1  0.196364  0.189405  0.437273
0.248309     120        1229
3           1  0.200000  0.212122  0.590435
0.160296     108        1454
4           1  0.226957  0.229270  0.436957
0.186900      82        1518

    cnt
0   985
1   801
2  1349
3  1562
4  1600
```

Data stats:

	instant	season	yr	mnth	holiday	weekday	orkingday	eathersit	temp	atemp	hum	ndspeed	casual	registered	cnt
count	731	731	731	731	731	731	731	731	731	731	731	731	731	731	731
mean	366	2.49	0.50	6.51	0.02	2.99	0.68	1.39	0.49	0.47	0.62	0.19	848	3656	4504
std	211	1.11	0.50	3.45	0.16	2.00	0.46	0.54	0.18	0.16	0.14	0.07	686	1560	1937
min	1.00	1.00	0.00	1.00	0.00	0.00	0.00	1.00	0.05	0.07	0.00	0.02	2.00	20.0	22.0
25%	183.5	2.00	0.00	4.00	0.00	1.00	0.00	1.00	0.33	0.33	0.52	0.13	315	2497	3152
50%	366	3.00	1.00	7.00	0.00	3.00	1.00	1.00	0.49	0.48	0.62	0.18	713	3662	4548
75%	548	3.00	1.00	10.0	0.00	5.00	1.00	2.00	0.65	0.60	0.73	0.23	1096	4776	5956
max	731	4.00	1.00	12.0	1.00	6.00	1.00	3.00	0.86	0.84	0.97	0.50	3410	6946	8714

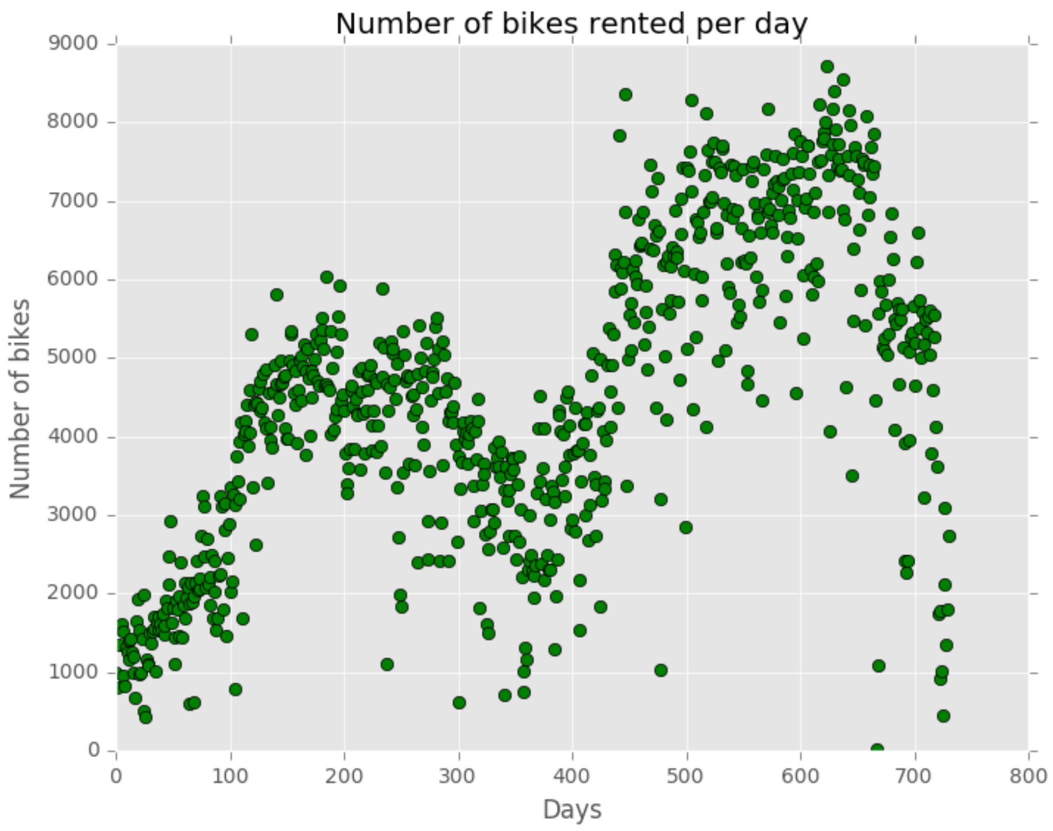
Characteristics

- The dataset is very concise and missing values are not a problem.
- Most of the data is already normalized or binary.

- Categorical data like 'weekday' or 'working day' are already processed.

2.2 Exploratory Visualization

The visualization shows a classic seasonal pattern with an up trend year over year. There are some outliers. These are left in the dataset because they are not due to measurement errors, but to extreme weather conditions. Because extreme weather conditions are part of the problem, so the data is not excluded.



2.3 Data Preprocessing (Methodology)

Dates get dropped because the regressor can not read this datatype and the order information is already stored in the index. The instant variable replicates this information also.

2.4 Algorithms and Techniques

Two types of regressors are trained. An SVR and a DNN-Regressor. Both are first used off-the-shelf with default parameters to create a benchmark.

> http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR
> https://github.com/tensorflow/skflow/blob/master/g3doc/api_docs/python/estimators.md

2.5 Benchmark

Both "benchmarks" for the coefficient of determination are very low. Parameter tuning is mandatory.

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
epsilon=0.1, gamma='auto',
  kernel='rbf', max_iter=-1, shrinking=True,
tol=0.001, verbose=False)
```

Score SVR:
```
-0.031042
```

```
skflow.TensorFlowDNNRegressor
(hidden_units=[10,10], steps=5000, learning_rate=0.1,
batch_size=1)
```

Score:
```
-0.082288
```

3 Methodology

3.1 Implementation

The regressors are trained using randomized search and cross-validation to identify the area of the best parameters. Then a grid search is used to tune parameter values of the regressor functions.

> http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html
> http://scikit-learn.org/stable/modules/generated sklearn.grid_search.RandomizedSearchCV.html

Tuning SVR with GridSearch

```
GridSearchCV(cv=None, error_score='raise',
       estimator=SVR(C=1, cache_size=200, coef0=0.0,
degree=3, epsilon=0.1, gamma='auto',
  kernel='rbf', max_iter=-1, shrinking=True,
tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid=[{'kernel': ['linear', 'rbf'], 'C':
[1000, 3000, 10000]}],
       pre_dispatch='2*n_jobs', refit=True,
scoring='r2', verbose=0)
```

Best parameter from grid search:

```
{'kernel': 'linear', 'C': 3000}
```

SVR Results

```
Score SVR: -0.031042
Score SVR tuned GS: 0.798317
```

SVR tuned with RandomizesSearch

best CV score from grid search:

```
0.769187
```

corresponding parameters:

```
{'kernel': 'linear', 'C': 10470.023139013298}
```

SVR Results

```
Score SVR: -0.031042
Score SVR tuned GS: 0.798317
Score SVR tuned RS: 0.802120
```

The tuning works for the SVR.

DNN-Regressor tuned with GS

best CV score from grid search:

```
0.197747
```

corresponding parameters:

```
 {'steps': 100, 'learning_rate': 1.0, 'hidden_units':
```

```
[13, 13], 'batch_size': 250}
```

Score:
```
0.138676
```

DNN Regressor Results

```
DNN: -0.082288
DNN tuned grid: 0.138676
```

DNN-Regressor tuned with RandomizesSearch

best CV score from grid search:
```
0.296782
```

corresponding parameters:
```
{'learning_rate': 0.7765638848002953, 'hidden_units':
[12, 12], 'batch_size': 318}
```

Score:
```
0.143749
```

Same picture with the DNN Regressor. The tuning helps, but the results are still underwhelming. Also, the best DNN result is no match for the tuned SVR.

Results

```
SVR: -0.031042
SVR tuned grid: 0.798317
SVR tuned random: 0.802120

DNN: -0.082288
DNN tuned grid: 0.138676
DNN tuned random: 0.143749
```

SVR works better than the DNN Regressor.

3.2 Refinement

The count of rented bikes (cnt) is just the sum of the features casual and registered. Two separate models are trained to predict these features. And add up afterward. This split should improve the projection.

SVR with GridSearch - for casual users

Feature columns:

```
Index([u'instant', u'dteday', u'season', u'yr',
u'mnth', u'holiday',
       u'weekday', u'workingday', u'weathersit',
u'temp', u'atemp', u'hum',
       u'windspeed'],
      dtype='object')
```

Target column:

```
casual
```

```
GridSearchCV(cv=None, error_score='raise',
       estimator=SVR(C=1, cache_size=200, coef0=0.0,
degree=3, epsilon=0.1, gamma='auto',
  kernel='rbf', max_iter=-1, shrinking=True,
tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid=[{'kernel': ['linear', 'rbf'], 'C':
[1, 3, 10, 30, 100, 300, 1000, 3000]}],
       pre_dispatch='2*n_jobs', refit=True,
scoring='r2', verbose=0)
```

Best parameter from grid search:

```
{'kernel': 'linear', 'C': 300}
```

SVR with RandomizesSearch - for casual users

best CV score from grid search:

```
0.652505
```

corresponding parameters:

```
{'kernel': 'linear', 'C': 3263.1463798458485}
```

SVR for casual with with GridSearch - for registered users

Feature column(s):

```
Index([u'instant', u'dteday', u'season', u'yr',
u'mnth', u'holiday',
       u'weekday', u'workingday', u'weathersit',
u'temp', u'atemp', u'hum',
       u'windspeed'],
      dtype='object')
```

Target column:

```
registered
```

```
GridSearchCV(cv=None, error_score='raise',
       estimator=SVR(C=1, cache_size=200, coef0=0.0,
degree=3, epsilon=0.1, gamma='auto',
  kernel='rbf', max_iter=-1, shrinking=True,
tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid=[{'kernel': ['linear', 'rbf'], 'C':
[1000, 3000, 10000]}],
       pre_dispatch='2*n_jobs', refit=True,
scoring='r2', verbose=0)
```

Best parameter from grid search:

```
{'kernel': 'linear', 'C': 1000}
```

SVR with RandomizesSearch - for registered users

best CV score from grid search:

```
0.652505
```

corresponding parameters:

```
{'kernel': 'linear', 'C': 3263.1463798458485}
```

4 Results

4.1 Model Evaluation and Validation

```
Score cas: 0.619215
Score reg: 0.774632
Score sum: 0.802635
```

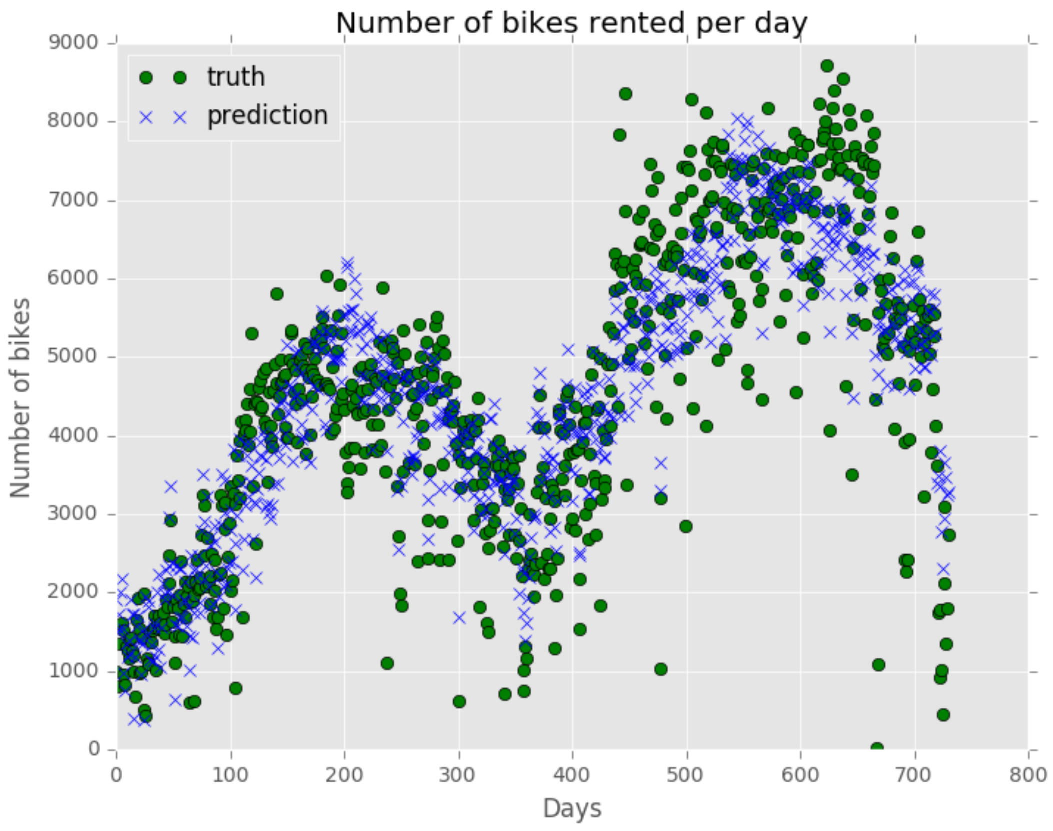
4.2 Justification

```
SVR: -0.031042
SVR tuned grid: 0.798317
SVR tuned RS: 0.802120

DNN: -0.082288
DNN tuned grid: 0.138676
DNN tuned random: 0.143749

SVR sum: 0.802635
```

- The SVR beats the DNN Regressor by far.
- The separate prediction of casual and registered customers increases the R^2 slightly.
- More than 80% determination is a decent result.

5 Conclusion

5.1 Free-Form Visualization



The predictions are reasonably good, without overfitting.

5.2 Reflection

- I had high hopes for the DNN Regressor. It was kind of disappointing that it does not even come close. Maybe my tuning was not right or it needs more data or computational power.
- Utilizing grid and randomize search in a way that makes sense was a little tricky. It makes more sense to start with a broad grid search and than use randomized search on the given interval, instead of vis a versa. It is also computational more efficient.

5.3 Improvement

- The determination could be increased by additional iterations in training and the number of folds in the cross-validation, at the expense of computing time.