Comparing Decision Trees with different maximum depths in estimating usage of Helsinki City Bikes

Introduction

Helsinki City Bikes are used by many citizens and an issue that users often come across is that some stations become empty, meaning no bikes are left to use. This is a problem especially in the times of day the bikes are used for commuting purposes, such as morning time.

In this report I aim to predict the number of departures from a certain bike station at a given time period of the day. I will compare two different machine learning methods for this problem. In the following section, the problem is formulated, the third section includes discussion on the data set and the used methods. Finally, the last two sections discuss the results and conclusions of the models' performance.

Problem formulation

The number of departures from a bike station at a certain time period may depend on the month of the year, whether it's a weekday or weekend or the air temperature during the specific time. The dataset used for this problem is downloaded from Kaggle [1] and contains rows representing all single trips taken with Helsinki City Bikes between 2016 and 2020. As I am interested in the total count of departures, the data is transformed so that data points represent days at a specific bike station, and the label represents the number of departures from this station at a given time period. Possible features could be day of the week, month, or air temperature.

Methods

The original dataset [1] contains more than 10 million rows, each representing a single trip. The columns include for instance the names of the departure and return stations, the departure and return times, duration of the trip, and air temperature at the time of the trip. In order to make the amount of data more manageable, I chose to take into consideration only one bike station (Kamppi) and a specific time period of the day (6:00-10:00).

The data set is transformed so that instead of having single trips as data points, the **data points represent a day at a specific bike station at 6:00 – 10:00**. I filtered out all other stations and departure times, and grouped the data by date, getting the average temperature of the given time period. The **label represents the count of departures**, and was calculated by, again, grouping the data by date, and getting the count of rows for each day at the given time period. I created new columns representing the day of week (0=Monday, …, 6=Sunday) and month of year, by extracting the values from the date column with pandas datetime functionalities. I created a function to determine whether the day is a workday or weekend

and created a new column with this information using the values of the days of week column. After processing the data, the dataset ended up with **984 data points**.

The possible **features** for the problem are:

- **workday** (int, 0 or 1): the bikes are often used for commuting purposes, and I noticed a significant difference in number of departures when comparing workdays and weekends
- **day of week** (int, 0 to 6): based on the data, on different days of week there seems to be significant differences in the usage of bikes
- **month of year** (int): based on the data, the bikes are used differently in summertime than in spring and fall
- **temperature** (int): intuitively, the air temperature may affect people's choices on using the bikes.

For the first model I chose to use the Decision Tree for Regression, since it's simple to understand and the model doesn't require a lot of data preparation [2]. Also, the features for this problem are non-linear, and these types of parameters don't affect the performance of a decision tree model. For the loss function I decided to use root mean squared error (RMSE), which is the standard deviation of the measures of how far from the regression line data points are.

$$RSME = [\frac{1}{n}\sum_{i=1}^{n}(y^i - h(x)^i)^2]^{1/2}$$

I chose this, because it's simple to understand and decision trees use squared error by default to measure the quality of the split [3]. I used different decision tree models with different maximum depths and saw that with maximum depth = 6 the model would produce the best validation error.

To split the data into training, validation and test sets, I used the sklearn train_test_split function, that splits the data randomly. The split was done so that training set contains 60% of data and validation and test sets each contain 20% of the data. I used this split, because I wanted to have enough data for training the model. Any larger amount than 60% of the data for the training set appeared to result in the model overfitting.

The result of using this model:

```
Maximum depth: 6
Training error: 17.46431450046308
Validation error: 17.753316315341557
```

I couldn't find another model that would give reasonable results for my data, so for the second model I chose to try the Decision Tree Regressor with a smaller maximum depth of 3, because with a too high depth, the decision tree might overfit the training data. I used the same split and loss function as for the first model.

The errors obtained from this model are:

```
Maximum depth: 3
Training error: 18.136749331462525
Validation error: 18.433157629416762
```

I also tried to cross validate the data using the sklearn KFold function for this model. I got the training error to decrease to 17.323, which is slightly better than the training error in the first model, but the validation error rose, so the original 60-20-20 split seems to work best.

Results

I used the sklearn R^2 regression score to measure accuracies of the two models and got the following results:

	max depth = 6 (R2 score)	max depth = 3 (R2 score)
Training	0.4786	0.4377
Validation	0.4623	0.4204

Based on the comparison between validation errors and the R^2 scores of the two models, the Decision Tree Regressor with maximum depth of 6 is the final chosen method. With both models, training and validation errors are very close to each other, which is desirable, but the first model simply gave a smaller validation error. I tested the model on my test set and obtained a test error of 18.518 and R^2 score of 0.4392, which are not significantly worse than those measured on the validation set (17.7533 and 0.4623).

Conclusion

I managed to find a model that is suitable for the type of data. However, the results are not good, and I suspect that my features do not correlate enough with the labels. Improvement could be done by getting more features from different data sources, for example precipitation could be a good indicator of the use of bikes. For example, I noticed that temperature has little effect on the labels, so I decided to drop the feature, and this gave me slightly better results. The problem is quite useful in real life and could be improved by doing better feature selection.

References

[1] https://www.kaggle.com/geometrein/helsinki-city-bikes

[2] https://scikit-learn.org/stable/modules/tree.html#tree

[3] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

ML project

April 3, 2022

[31]:
```
import numpy as np
import pandas as pd
```

[32]:
```
df = pd.read_csv('database.csv', usecols=['departure', 'departure_name', 'Air␣
↪temperature (degC)'])
df.head(5)
```

[32]:

	departure	departure_name	Air temperature (degC)
0	2020-03-23 06:09:44	Kuusitie	0.9
1	2020-03-23 06:11:58	Kamppi (M)	0.9
2	2020-03-23 06:16:29	Porolahden koulu	0.9
3	2020-03-23 06:33:53	Vallipolku	0.9
4	2020-03-23 06:36:09	Länsisatamankatu	0.9

[33]:
```
data = df[df['departure_name'] == 'Kamppi (M)']
data[['date', 'time']] = data['departure'].str.split(' ', expand=True)
data = data.drop(['departure', 'departure_name'], axis=1)
print(data.shape)
data.head(5)
```

```
(201560, 3)
```

```
/opt/conda/lib/python3.8/site-packages/pandas/core/frame.py:3069:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[k1] = value[k2]
```

[33]:

	Air temperature (degC)	date	time
1	0.9	2020-03-23	06:11:58
114	1.5	2020-03-23	08:20:34
323	2.1	2020-03-23	10:47:38
367	2.7	2020-03-23	11:19:47
376	2.7	2020-03-23	11:22:51

[34]:
```
data.columns = ['temperature', 'date', 'time']
data[['year', 'month', 'day']] = data['date'].str.split('-', expand=True)
data = data.drop('year', axis=1)
data['time window'] = data['time'].apply(lambda x: int(x[0:2]))
data.head(5)
```

[34]:

	temperature	date	time	month	day	time window
1	0.9	2020-03-23	06:11:58	03	23	6
114	1.5	2020-03-23	08:20:34	03	23	8
323	2.1	2020-03-23	10:47:38	03	23	10
367	2.7	2020-03-23	11:19:47	03	23	11
376	2.7	2020-03-23	11:22:51	03	23	11

[35]:
```
newdata = data[(data['time window'] <= 9) & (data['time window'] >= 6)]
newdata = newdata.drop('time window', axis=1)

newdata.head()
```

[35]:

	temperature	date	time	month	day
1	0.9	2020-03-23	06:11:58	03	23
114	1.5	2020-03-23	08:20:34	03	23
2603	2.7	2020-03-24	06:19:45	03	24
2845	3.5	2020-03-24	08:32:01	03	24
2860	3.5	2020-03-24	08:42:13	03	24

[36]:
```
grouped_data = newdata.groupby(by=['date'],as_index=False).mean()
grouped_data['date'] = pd.to_datetime(grouped_data['date'])
grouped_data['dayofweek']=grouped_data['date'].dt.dayofweek
grouped_data['monthofyear'] = pd.DatetimeIndex(grouped_data['date']).month
grouped_data['count'] = newdata.groupby(by=['date'],as_index=False).
↪count()['time']

grouped_data.head(20)
```

[36]:

	date	temperature	dayofweek	monthofyear	count
0	2016-05-03	13.420000	1	5	10
1	2016-05-04	14.281818	2	5	11
2	2016-05-05	17.200000	3	5	1
3	2016-05-06	16.211111	4	5	9
4	2016-05-09	16.471429	0	5	14
5	2016-05-10	16.748148	1	5	27
6	2016-05-11	11.566667	2	5	15
7	2016-05-12	12.452381	3	5	21
8	2016-05-13	12.285714	4	5	21
9	2016-05-14	12.900000	5	5	1
10	2016-05-15	11.625000	6	5	4
11	2016-05-16	9.442857	0	5	7

```
12 2016-05-17    10.010000          1             5      20
13 2016-05-18    11.748387          2             5      31
14 2016-05-19    11.517647          3             5      17
15 2016-05-20    13.341667          4             5      24
16 2016-05-21    15.666667          5             5       3
17 2016-05-22    11.400000          6             5       1
18 2016-05-23    17.416000          0             5      25
19 2016-05-24    18.357692          1             5      26
```

[37]:
```
def weekday (x):
    if x < 5:
        return 1
    else:
        return 0
grouped_data['weekday'] = grouped_data['dayofweek'].apply(lambda x: weekday(x))
grouped_data.head(10)
```

[37]:

	date	temperature	dayofweek	monthofyear	count	weekday
0	2016-05-03	13.420000	1	5	10	1
1	2016-05-04	14.281818	2	5	11	1
2	2016-05-05	17.200000	3	5	1	1
3	2016-05-06	16.211111	4	5	9	1
4	2016-05-09	16.471429	0	5	14	1
5	2016-05-10	16.748148	1	5	27	1
6	2016-05-11	11.566667	2	5	15	1
7	2016-05-12	12.452381	3	5	21	1
8	2016-05-13	12.285714	4	5	21	1
9	2016-05-14	12.900000	5	5	1	0

[38]:
```
y = grouped_data['count'].to_numpy()
X = grouped_data.drop(['date', 'count', 'temperature'], axis=1)
print(X.shape)
print(y.shape)
```

```
(984, 3)
(984,)
```

[39]:
```
from sklearn.model_selection import train_test_split
train_ratio = 0.6
validation_ratio = 0.2
test_ratio = 0.2

X_train, X_rem, y_train, y_rem = train_test_split(X,y,test_size=1 -␣
 ↪train_ratio,random_state=42)
```

```
X_val, X_test, y_val, y_test =␣
 ↪train_test_split(X_rem,y_rem,test_size=test_ratio/(test_ratio +␣
 ↪validation_ratio),random_state=42)

print(X_train.shape, X_val.shape, X_test.shape)
```

```
(590, 3) (197, 3) (197, 3)
```

[40]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, accuracy_score, r2_score
from sklearn import tree
```

[41]:
```
dt_regr_1 = DecisionTreeRegressor(max_depth=6, random_state=3)
dt_regr_1.fit(X_train, y_train)

y_pred_train = dt_regr_1.predict(X_train)
y_pred_val = dt_regr_1.predict(X_val)

tr_error = mean_squared_error(y_train, y_pred_train)
val_error = mean_squared_error(y_val, y_pred_val)
rmse_dt_tr = tr_error**(1/2)
rmse_dt_val = val_error**(1/2)
print('Maximum depth:', dt_regr_1.get_depth())
print('Training error:', rmse_dt_tr)
print('Validation error:', rmse_dt_val)
```

```
Maximum depth: 6
Training error: 17.46431450046308
Validation error: 17.753316315341557
```

[42]:
```
r2_train = r2_score(y_train, y_pred_train)
r2_val = r2_score(y_val, y_pred_val)
print(r2_train)
print(r2_val)
```

```
0.4786067752869113
0.4623463623422036
```

[43]:
```
dt_regr_2 = DecisionTreeRegressor(max_depth=3, random_state=3)
dt_regr_2.fit(X_train, y_train)

y_pred_train = dt_regr_2.predict(X_train)
y_pred_val = dt_regr_2.predict(X_val)

tr_error = mean_squared_error(y_train, y_pred_train)
val_error = mean_squared_error(y_val, y_pred_val)
```

```
rmse_dt_tr = tr_error**(1/2)
rmse_dt_val = val_error**(1/2)
print('Maximum depth:', dt_regr_2.get_depth())
print('Training error:', rmse_dt_tr)
print('Validation error:', rmse_dt_val)
```

```
Maximum depth: 3
Training error: 18.136749331462525
Validation error: 18.433157629416762
```

[44]:
```
r2_train_2 = r2_score(y_train, y_pred_train)
r2_val_2 = r2_score(y_val, y_pred_val)
```

[45]:
```
print(r2_train_2)
print(r2_val_2)
```

```
0.43768302056973185
0.42038037681509544
```

[46]:
```
dt_regr_1.fit(X_train, y_train)
y_pred_test = dt_regr_1.predict(X_test)

test_error = mean_squared_error(y_test, y_pred_test)
rmse_dt_test = test_error**(1/2)
print('Maximum depth:', dt_regr_1.get_depth())
print('Test error:', rmse_dt_test)
```

```
Maximum depth: 6
Test error: 18.518303930341634
```

[47]:
```
r2_score(y_test, y_pred_test)
```

[47]:
```
0.4391676373724376
```

[48]:
```
from sklearn.model_selection import train_test_split, KFold

cv = KFold(n_splits=3)

tr_errors = []
val_errors = []

for train_index, val_index in cv.split(y):

    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
    y_train, y_val = y[train_index], y[val_index]

    dt_regr_2 = DecisionTreeRegressor(max_depth=3, random_state=3)
```

```
    dt_regr_2.fit(X_train, y_train)

    y_pred_train = dt_regr_2.predict(X_train)
    y_pred_val = dt_regr_2.predict(X_val)

    tr_error = mean_squared_error(y_train, y_pred_train)
    val_error = mean_squared_error(y_val, y_pred_val)
    rmse_dt_tr = tr_error**(1/2)
    rmse_dt_val = val_error**(1/2)

    tr_errors.append(rmse_dt_tr)
    val_errors.append(rmse_dt_val)

train = sum(tr_errors)/len(tr_errors)
val = sum(val_errors)/len(val_errors)
print('Training error:', train)
print('Validation error:', val)
```

```
Training error: 17.3228048151921
Validation error: 22.252388104703215
```

[ ]: