Availability of Helsinki city bikes at Aalto-university (M) analysis with different regression methods

March 2022

1 Introduction

Helsinki City bikes is a public bicycle system in Helsinki and Espoo, and it is part of the public transport in Helsinki. There are 347 stations around Helsinki and Espoo and 3480 bikes in use (HSL). In 2019 city bike systems had over 60 000 registered seasonal users and there were 3.7 million trips cycled (Sweco).

This analysis and machine learning model should be useful in predicting the numbers of available bikes at a certain time at Aalto-University (M) bike station during first 2 weeks of the season. This should help the Helsinki Regional Transport Authority (HSL), owner and operator of the system, in predicting times when bikes need to be transferred to (or from) Aalto-University (M) bike station to ensure availability of bikes around the clock. HSL's route planner can use the data in planning trips and whole public transport.

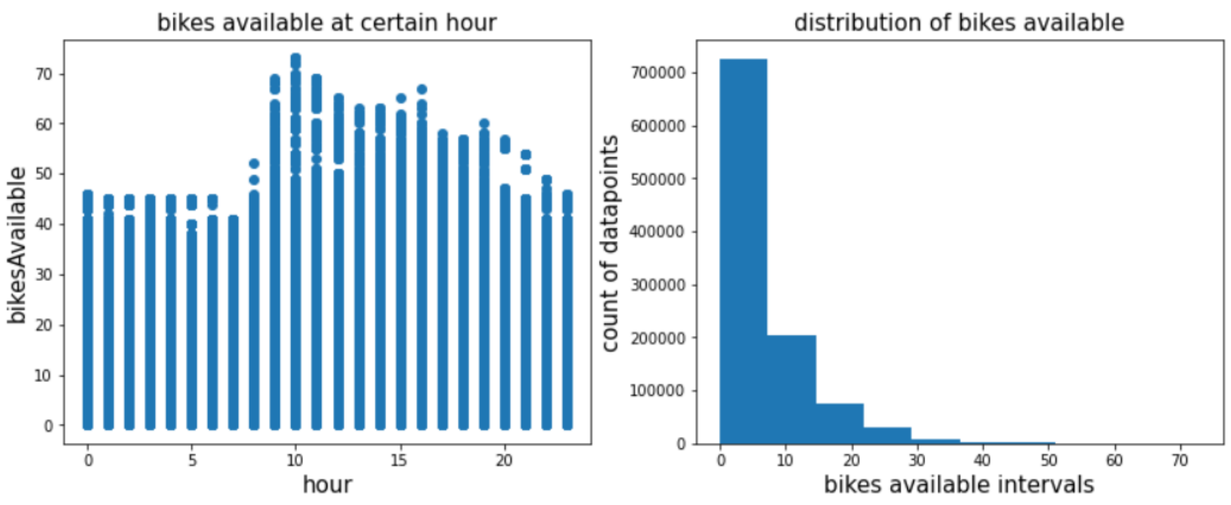
The machine learning problem is explained in more detail in the Problem Formulation section. Description and preprocessing of the data and regression methods are described in the Methods section. Discussion of the results are in the Results section. Summary of the problem, reflection of the results and discussion of improvements are in the Conclusion section. There are references and the code at the end of the report.

2 Problem Formulation

The problem at hand: how many city bikes there are at Aalto-University (M) station at a certain hour.

The data points of the problem are the number of available bikes at Aalto-University (M) city bike station every five seconds. Data is obtained from avoindata.fi (avoindata.fi), which is open data collection of numerous Finnish data. Each data point is a measurement of available bikes in the station, labelled with station's id and time information. Number of available bikes and date-time information, both integers, are used in this analysis.

In short, the **label** in the problem is number of available city bikes at the Aalto-University (M) station (ranging from 0 to 70). **Features** are hours of the whole day (00-23).



Screenshot 1: Plots of the data

3 Methods

The data set obtained from avoindata.fi for city bike’s usage in 2019 has 25 966 001 rows of data. To limit data points and therefore computational time needed for analysis, I’ve decided to use only the April of 2019 season for this analysis.

As stated in the section Problem Formulation, each data point is an integer measurement for available bikes and free slots in the station, station’s id and datetime. Unnecessary columns about the bike station and datetime were removed (code cell 2) to reduce data’s memory usage.

Feature selection was straightforward from the problem formulation. Hours were chosen as features instead of minutes (or seconds) to achieve desired accuracy and outcome for the problem. There was also intention to keep number of features relatively restrained. This affects the results of this report, more about in section conclusion.

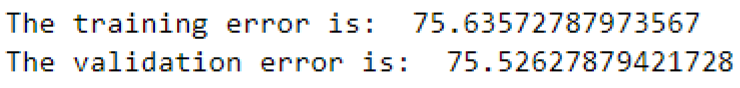
The first model chosen for this problem was linear regression. Decision was made because I believed problem to be somewhat linear for a single day period. I pictured data to behave as follows: “more (or less) bikes in the night and morning, less (or more) bikes in the day and evening”. After applying linear regression and doing some plotting, method turned out to be suboptimal. More about it in the Results section. The loss function used was mean squared error loss because the data seemed to have few outliers. This assumption was proven to be totally wrong, more about it in the section Results.

The second model chosen was polynomial regression. I chose it since the data seemed to be somewhat traceable using polynomials of higher degree. Even though data points were visually evenly distributed over most of the labels and features (see Screenshot 1), points weren’t weighing as evenly. The intervals between the numbers of bicycles varied during the day, which is not seen from the plotting. The loss function used was once again mean squared error loss. Decision to use the same loss function was made to compare results of two regressions.

The data was split into training set, validation set and test set. Splitting was done randomly using function train_test_split() provided by Scikit-learn's model_selection module two times. First into train and remaining groups in ratio 50:50. Then the remaining group is split into validation and test in ratio 1:5. Design decision to choose such ratios were motivated by the course's lecture materials. Since the raw data is enormous and evenly distributed throughout the whole set, ratios can be chosen freely.

4 Results

Using linear regression, training and validation errors were 75.635 and 75.526. Error values are absurdly high due to very large amount of data points, even distribution of data points in non-linear fashion, too few features chosen and similar behavior of data in training and validation sets. The value indicates the inadequacy of the chosen method.

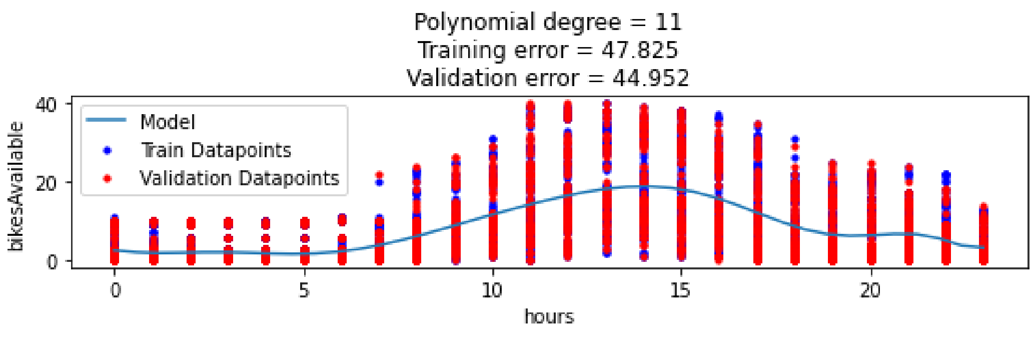


```
The training error is:   75.63572787973567
The validation error is:   75.52627879421728
```

Screenshot 2: Results of linear regression's training and validation errors.

Using polynomial regression, training errors ranged from 47.89 to 60.71. Validation error ranged from 44.95 to 58.85. The model performed better using larger polynomial degrees. Best results were obtained at relatively high degrees, probably due to the diffused distribution of weights over data points.

The final chosen method is polynomial regression using polynomial degree 11, since error values were the smallest. Training error was 30.6 and validation error 28.8. The polynomial degree of 11 is large enough to cover the scatter of data points, hence lower loss values. However, degrees over 11 had higher loss values, which may mean that the result obtained is a bit random.



Screenshot 3: Final chosen method.

Construction of the test set is described in the section Methods. The test error obtained is greater that our final method's: 49.72. Therefore, the final chosen method is not convincing, since the test error value is approximately the same as the errors obtained using numerous polynomial degrees (See sec. 7 Appendices). More review of the result in the next section.

```
The test error is:   49.71904956377122
```

Screenshot 4: The test error.

5 Conclusion

This report analyzed the number of HSL city bikes at Aalto-University (M) station during the first month of the season, April 2019. The number of bikes available was used as a label and an hour of the day as a feature. Vastness of data points and naive approach to the problem complicated the analysis. Error values are uncomfortably large.

As seen from the plot of the best obtained model, there is clear pattern of varying number of bikes at the station. Judging from gathered hypothesis one can expect up to 15 bikes at the station from 10 AM to 6 PM and less than 10 bikes from 11 PM to 6 AM. The bike operator should therefore fill the station with bikes at 6 PM to ensure availability of bikes throughout the day during the first month of the season to keep customers satisfied.

Limitation of linear regression is its linearity. Once the problem is non-linear, the method doesn't fit. Polynomial regression did well, yet it could not consider and model the data to the full, resulting in high error values. The data's behavior and distribution seem to be hard for any machine learning method to predict. More complex machine learning models could improve the result.

This report is only indicative predicting tool for the problem. The analysis could be improved by using each minute of the day as features instead of hours. Result would be more precise, since the ratio between the number of labels and features would be more optimal. Improvement wasn't implemented due to lack of time. Too great error values and methods bad performance are due to this wrong feature selection.

The chosen regression methods don't give the best results for this problem. Huber regression or ridge regression methods and loss functions could have provided better results, since the data consist of several outliers. The results would have been more detailed if these proposed improvements were made. The nature of the solution would otherwise be similar.

To make the analysis "commercial level", should the analysis be more comprehensive in terms of factors affecting the use of city bikes, like the weather. Improvements mentioned above would be a good starting point, but other machine learning methods should also be explored. The data of users could also be obtained and used for similar purposes.

6 References

Sweco: https://www.sweco.fi/projektit/kaupunkipyorailyn-suosio-jatkaa-voittokulkuaan/

City of Helsinki: https://liikkumisvahti.hel.fi/

avoindata.fi: https://www.avoindata.fi/data/fi/dataset/kaupunkifillareiden-telinekohtaiset-vapaiden-pyorien-maarat-kausilta-2017-2018-ja-2019

7 Appendices

stage 3

April 3, 2022

[1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

[ ]:
```
# read data from file
df = pd.read_csv('whole_data_2019.csv', sep=";")

# get rid of unnecessary data. datetime dropped to reduce data file size.
df = df.
 ↪drop(['spacesAvailable','allowDropoff','isFloatingBike','state','realTimeData','week','yday
 ↪axis=1)

# df.info()
# RangeIndex: 25966000 entries, 0 to 25965999
# Data columns (total 4 columns):
#  #   Column          Dtype
#  ---  -------         -----
#  0   id              int64
#  1   bikesAvailable  int64
#  2   time            object
#  3   hour            int64
#  4   month           int64
#  5   hour            int64
#  6   year            int64
# dtypes: int64(6), object(1)
# memory usage: 1.4+ GB
```

[ ]:
```
# obtain data for Aalto-yliopisto (M) bike station for April.
df = df.loc[df["id"] == 541]
data = df.loc[df["month"] == 4]



# define labels and features
```

```
X = data["hour"]
y = data["bikesAvailable"]

X = X.to_numpy()
y = y.to_numpy()
X = X.reshape(-1, 1)
```

[ ]:
```
# plot the data
fig, axes = plt.subplots(1, 2, figsize=(14,5)) # create a figure with two axes␣
 ↪(1 row,2 columns) on it
axes[0].scatter(data['hour'],data['bikesAvailable']) # plot a scatter plot on␣
 ↪axes[0] to show the relation between MinTemp and MaxTemp
axes[0].set_xlabel("hour",size=15)
axes[0].set_ylabel("bikesAvailable",size=15)
axes[0].set_title("hour vs bikesAvailable ",size=15)

axes[1].hist(data['bikesAvailable']) # plot a hist plot to show the␣
 ↪distribution of MaxTemp
axes[1].set_title('distribution of bikesAvailable',size=15)
axes[1].set_ylabel("count of datapoints",size=15)
axes[1].set_xlabel("bikesAvailable intervals",size=15)
plt.show()
```

[ ]:
```
## Split the dataset into a training set and a remaining set with␣
 ↪train_test_split.
## The test_size=0.5 and random_state=42:
X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.5,␣
 ↪random_state=42)

## Split the remaining dataset into a validation set and test set with␣
 ↪train_test_split.
## The test_size=0.2 and random_state=42:
X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size=0.2,␣
 ↪random_state=42)
```

[ ]:
```
# Linear Regression and plot
regr = LinearRegression().fit(X_train, y_train)

# predict label values based on features and calculate the training error
y_train_pred = regr.predict(X_train)
train_tr_error = mean_squared_error(y_train, y_train_pred)

y_val_pred = regr.predict(X_val)
val_error = mean_squared_error(y_val, y_val_pred)

print('The training error is: ', train_tr_error)
print('The validation error is: ', val_error)

# visualize linear regression model
plt.figure(figsize=(10, 8))    # create a new figure with size 10*8

plt.scatter(X_train, y_train, color="blue", s=15, label="Train Datapoints")
plt.scatter(X_val, y_val, color="black", s=15, label="Validation Datapoints")

# plot the predictions obtained by the learnt linear hypothesis
plt.plot(X_train, y_train_pred, color='r', label='h(x)')

plt.xlabel('hours',size=15) # define label for the horizontal axis
plt.ylabel('bikesAvailable',size=15) # define label for the vertical axis

plt.title('Linear regression model',size=15) # define the title of the plot
plt.legend(loc='best',fontsize=10) # define the location of the legend

plt.show()  # display the plot on the screen
```

[ ]:
```
# Polynomial regression and plots
degrees = [2, 3, 6, 8, 9, 10, 11, 12]

# declare a variable to store the resulting training errors for each polynomial␣
 ↪degree
tr_errors = []
val_errors = []

plt.figure(figsize=(8, 20))
for i, degree in enumerate(degrees):    # use for-loop to fit polynomial␣
 ↪regression models with different degrees
    plt.subplot(len(degrees), 1, i + 1)    # choose the subplot

    lin_regr = LinearRegression(fit_intercept=False) # NOTE:␣
 ↪"fit_intercept=False" as we already have a constant iterm in the new feature␣
 ↪X_poly

    poly = PolynomialFeatures(degree=degree)    # generate polynomial features
    X_train_poly = poly.fit_transform(X_train)    # fit and transform the raw␣
 ↪features
    lin_regr.fit(X_train_poly, y_train)    # apply linear regression to these␣
 ↪new features and labels

    y_pred_train = lin_regr.predict(X_train_poly)    # predict using the linear␣
 ↪model
    tr_error = mean_squared_error(y_train, y_pred_train)    # calculate the␣
 ↪training error
```

```
    X_val_poly = poly.transform(X_val) # transform the raw features for the␣
 ↪validation data
    y_pred_val = lin_regr.predict(X_val_poly) # predict values for the␣
 ↪validation data using the linear model
    val_error = mean_squared_error(y_val, y_pred_val) # calculate the␣
 ↪validation error

    tr_errors.append(tr_error)
    val_errors.append(val_error)

    X_fit = np.linspace(0, 23, 40)     # generate samples
    plt.tight_layout()
    plt.plot(X_fit, lin_regr.predict(poly.transform(X_fit.reshape(-1, 1))),␣
 ↪label="Model")     # plot the polynomial regression model
    plt.scatter(X_train, y_train, color="b", s=10, label="Train Datapoints")   ␣
 ↪# plot a scatter plot of y(maxtmp) vs. X(mintmp) with color 'blue' and size␣
 ↪'10'
    plt.scatter(X_val, y_val, color="r", s=10, label="Validation Datapoints")  ␣
 ↪ # do the same for validation data with color 'red'
    plt.xlabel('hours')    # set the label for the x/y-axis
    plt.ylabel('bikesAvailable')
    plt.legend(loc="best")    # set the location of the legend
    plt.title(f'Polynomial degree = {degree}\nTraining error = {tr_error:.
 ↪5}\nValidation error = {val_error:.5}')    # set the title

plt.show()    # show the plot
```

[ ]:
```
# Obtaining final test error
lin_regr = LinearRegression(fit_intercept=False) # NOTE: "fit_intercept=False"␣
 ↪as we already have a constant iterm in the new feature X_poly

poly = PolynomialFeatures(degree=11)    # generate polynomial features with␣
 ↪degree of 5
X_train_poly = poly.fit_transform(X_train)    # fit the raw features
lin_regr.fit(X_train_poly, y_train)    # apply linear regression to these new␣
 ↪features and labels


X_test_poly = poly.fit_transform(X_test)
y_pred_test = lin_regr.predict(X_test_poly)
test_error = mean_squared_error(y_test, y_pred_test)

print("The test error is: ",test_error)
```