Comparing Deep learning and Polynomial regression models for choosing a marketing strategy

Introduction

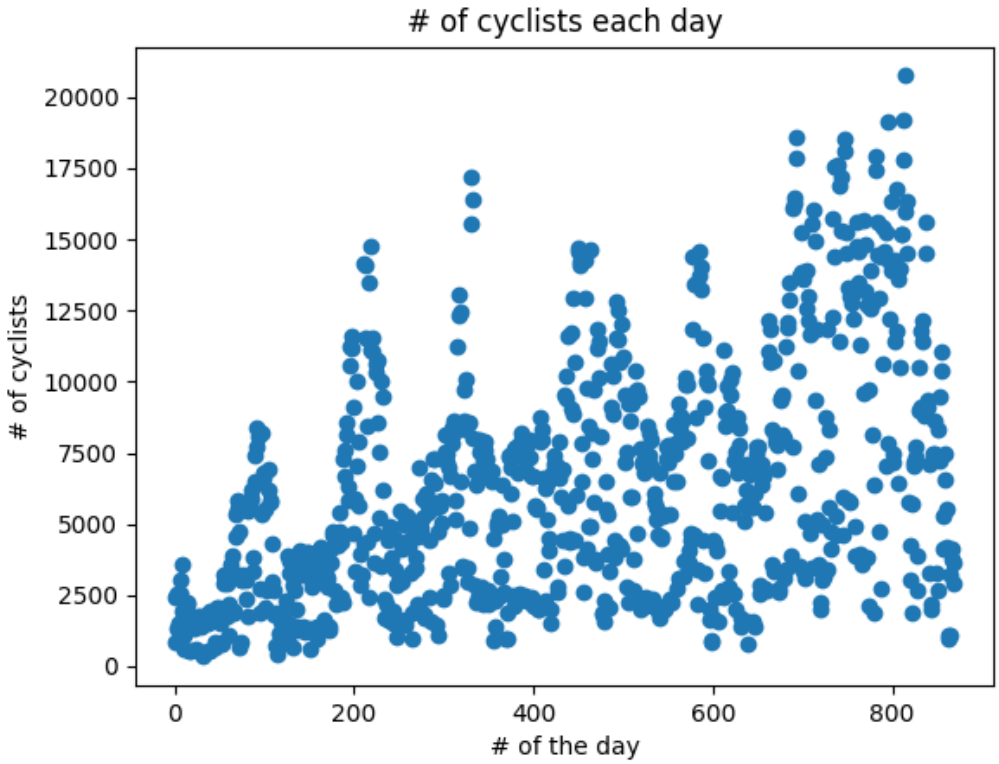
Bicycles are become more and more popular transport among the European citizens and in Finland in particular. With the popularity growth the market grows as well. More and more bicycle repair shops are set up. Imagine such a new company, which is going to show its advertisement on TV. The main goal of the advertisement is to attract as many customers as the repair shop can serve. This company discovered that the best effect from the TV advertisement is achieved in the days, when the number of cyclists on the streets as large as possible. Also, the winter/early spring maintenance is the most valuable for the company due to the heavy snow and an extreme low temperature.

According to this need the goal is to determine the day with the heaviest cycling traffic during the next winter/early spring period. The prediction of the number of cyclists on the Helsinki roads next winter/early spring will be made with the models described in the section "Methods". The dataset is explained in the section "Problem formulation". Section "Results" will provide the most appropriate model for this problem and show the errors in several predictions. In "Conclusion" section the best result is discussed and criticized. Finally, the code is provided in the end.

Problem formulation

The application of this problem can be seen in almost every marketing strategy.

The data points contain information about the hourly number of cyclists from 2014 to 2021 in 16 different points in Helsinki. The dataset was downloaded from the Helsinki region infoshare [1]. The time information will be reduced to the winter period only (from 01.12 to 31.03) and combined from hourly to daily intervals. These dates will be converted to the numeric range, since some methods do not support date format. The labels will represent the total number of cyclists each day from 01.12 to 31.03 in Helsinki. There is no missing features and labels since the label is the sum of the 16 measurements and at least 1 measurement is always present. After these manipulations the features will be represented by the number of the day and the labels will be represented by the total number of cyclists each day.



Methods

The data set was collected from Helsinki region infoshare[1]. There are number of years * number of days datapoints (31*8 + 30*8 + 27*6 + 28*2 + 31*8 = 954 datapoints). There is no missing data in any fields as described in the section “Problem formulation”. The “Cyclists” column will be labels and the “Date” column will be features.

People use bicycles mostly as a transport; therefore, the working days and weekends are strongly related to the number of cyclists on the roads. Average weather conditions are also included in the information about the date, since 8 years are considered. Even extreme weather conditions on some days are smoothen by taking 8 years into account. Therefore, 2 main parameters: working or rest day and weather condition are considered in the “Date” features. Moreover, the growing popularity of cycling is also considered.

954 datapoints is a relatively small dataset; therefore, a single split into training and validation sets can be very unlucky and does not represent the model’s performance. K-fold cross validation will determine the average validation error over all k folds and show the true model’s performance. We will split the data into 5 folds to achieve approximately 80:20 train/validation split. This ratio is one of the most popular and it can be further changed if needed.

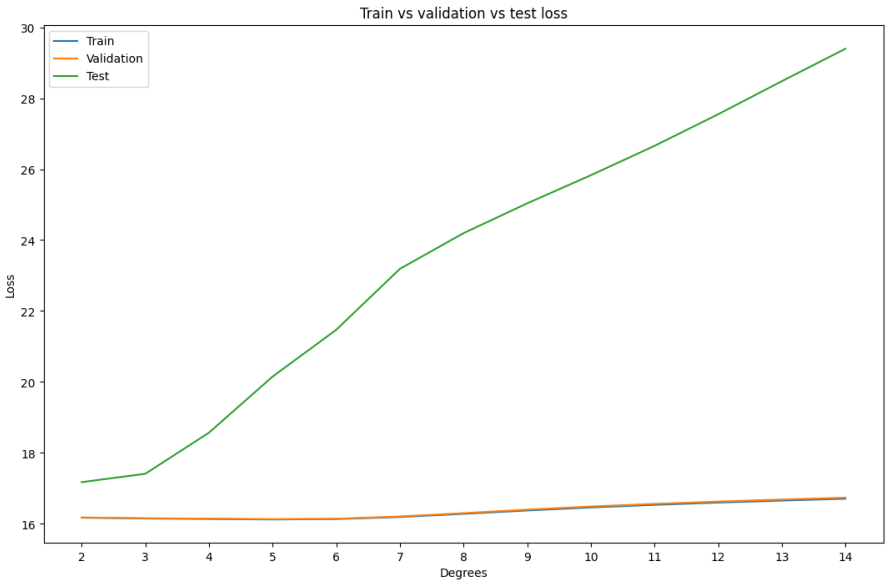
To show the relationship between features and labels the polynomial regression is used firstly. This model is used for ML problems involving datapoints with the single numeric feature (# of the day). Regular linear regression will not show a great performance, since it will ignore many hidden parameters in the data (weather conditions, holydays). The hypothesis space consists of polynomial maps of degree ***n***.

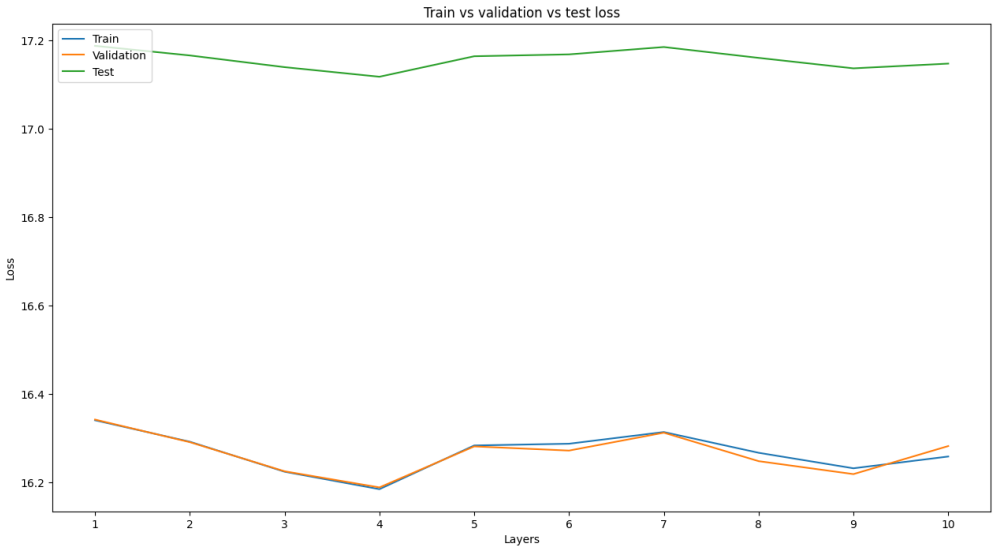
Multi-layer Perceptron model with 20 neurons on each layer was chosen as a second model for this problem. The activation function is ReLU (rectified linear unit) by default. This model will

show a great performance, since it is a very powerful model, which can approximate all continuous piecewise linear functions. The ANN defines a hypothesis space represented by all maps $h^{(\boldsymbol{w})}$ for different weights $\boldsymbol{w}$.

Mean squared error (MSE) is used as a loss function for both models. There are no notable outliers in the data, so MSE will work great here. Moreover, it is a convex and differentiable function, which will allow us to minimize it efficiently.

Results





The log function is implemented to the loss to make the values more readable.

Last 152 datapoints were chosen for testing. Almost 7 seasons from December to March are used for training and validation and 1 full season with an extra December month is chosen for testing. We consider all the hidden parameters such as weather conditions and weekends in the datapoints of the testing set.

Polynomial regression model from the hypothesis space of polynomials of degree 5 shows its best performance. The overfitting is observed starting with degree 6. Nevertheless, the test loss grows linearly.

Multi-layer Perceptron model shows much better results. The best training and validation losses can be observed in ANN with 4 hidden layers 20 neurons each. The test loss is almost constant with increasing number of hidden layers; nevertheless, the best test loss appears in ANN with 4 hidden layers.

Multi-layer Perceptron model shows better training and validation losses, but they similar to the polynomial regression model. So, both models were tested with the testing set. MLP model has much smaller test loss (17.0 – 17.2), than the polynomial regression model (17.0 – 30.0). Moreover, MLP model's test loss is close to the train and validation losses. That means, that MLP model is suitable for this problem.

Conclusion

In conclusion we can state, that MLP model perfectly fits the problem and can be chosen for the marketing strategy described in the introduction. The ways we can improve the performance:

1. Collect more data about the quantity of cyclists on the roads
2. Adding more features such as weather conditions, fuel prices, daily surveys. The MLP accepts multiple features, so this approach will improve the performance of the problem.

Testing other models is redundant, since the MLP is powerful enough. The dataset is relatively small to avoid problems with MLP computational power demands.

References

[1] https://hri.fi/data/fi/dataset/helsingin-pyorailijamaarat

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split, KFold
import math

# Data preparation and visualisation
data = pd.read_csv('Dataset.csv').fillna(0)

data["Cyclists"] = data.sum(axis=1)
data = data.drop(['Auroransilta', 'Eteläesplanadi', 'Huopalahti (asema)',
'Kaisaniemi/Eläintarhanlahti',
                  'Kaivokatu', 'Kulosaaren silta et.', 'Kulosaaren silta po.
', 'Kuusisaarentie',
                  'Käpylä - Pohjoisbaana', 'Lauttasaaren silta eteläpuoli',
'Merikannontie',
                  'Munkkiniemen silta eteläpuoli', 'Munkkiniemi silta
pohjoispuoli', 'Heperian puisto/Ooppera',
                  'Pitkäsilta itäpuoli', 'Pitkäsilta länsipuoli',
'Lauttasaaren silta pohjoispuoli',
                  'Ratapihantie', 'Viikintie', 'Baana'], axis=1)
data = data.rename(columns={"Päivämäärä": "Date"})

data = data.groupby(data.index // 24).sum()

# 0 - 89 are 01.01.2014 to 31.03.2014
# 334 - 454 are 01.12.2014 to 31.03.2015
# 699 - 820 are 01.12.2015 to 31.03.2016
# 1065 - 1185 are 01.12.2016 to 31.03.2017
# 1430 - 1550 are 01.12.2017 to 31.03.2018
# 1795 - 1915 are 01.12.2018 to 31.03.2019
# 2160 - 2281 are 01.12.2019 to 31.03.2020
# 2526 - 2646 are 01.12.2020 to 31.03.2021
# 2891 - 2921 are 01.12.2021 to 31.12.2021

idx = list(range(90, 334)) + list(range(455, 699)) + list(range(821, 1065)) +
list(
    range(1186, 1430)) + list(range(1551, 1795)) + list(range(1916, 2160)) +
list(range(2282, 2626)) + list(
    range(2647, 2891))
data = data.drop(idx).reset_index(drop=True)

test_set = data.tail(152)
X_test = test_set.index.to_numpy().reshape(-1, 1)
y_test = test_set["Cyclists"].to_numpy()
data.drop(data.tail(152).index, inplace=True)

X = data.index.to_numpy().reshape(-1, 1)
y = data["Cyclists"].to_numpy()
```

```
fig = plt.figure()

ax = fig.add_subplot(1, 1, 1)

ax.scatter(X, y)
ax.set_xlabel("# of the day")
ax.set_ylabel("# of cyclists")
ax.set_title("# of cyclists each day")

plt.show()

# Training/validation
cv = KFold(n_splits=5, random_state=42, shuffle=True)

# Polynomial regression
degrees = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
poly_train_errors = []
poly_val_errors = []
poly_test_errors = []

for deg in degrees:
    train_errors = []
    val_errors = []
    test_errors = []

    for train_index, val_index in cv.split(y):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        poly = PolynomialFeatures(degree=deg)
        X_train_poly = poly.fit_transform(X_train)

        lin_regr = LinearRegression(fit_intercept=False)
        lin_regr.fit(X_train_poly, y_train)

        y_pred_train = lin_regr.predict(X_train_poly)
        train_error = mean_squared_error(y_train, y_pred_train)
        train_errors.append(train_error)

        X_val_poly = poly.fit_transform(X_val)
        y_pred_val = lin_regr.predict(X_val_poly)
        val_error = mean_squared_error(y_val, y_pred_val)
        val_errors.append(val_error)

        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)
        test_errors.append(test_error)

    avg_train_error = math.log(sum(train_errors) / len(train_errors))
    avg_val_error = math.log(sum(val_errors) / len(val_errors))
    avg_test_error = math.log(sum(test_errors) / len(test_errors))
    poly_train_errors.append(avg_train_error)
    poly_val_errors.append(avg_val_error)
    poly_test_errors.append(avg_test_error)

plt.figure(figsize=(13, 30))

plt.plot(degrees, poly_train_errors, label='Train')
plt.plot(degrees, poly_val_errors, label='Validation')
plt.plot(degrees, poly_test_errors, label='Test')
plt.xticks(degrees)
plt.legend(loc='upper left')
```

```
plt.xlabel('Degrees')
plt.ylabel('Loss')
plt.title('Train vs validation vs test loss')
plt.show()

# for i, deg in enumerate(degrees):
#     print("Polynomial regression with degree ", deg, " has training error
", math.log(poly_train_errors[i]), "\n",
#           "Polynomial regression with degree ", deg, " has validation error
", math.log(poly_val_errors[i]), "\n",
#           "Polynomial regression with degree ", deg, " has test error ",
math.log(poly_test_errors[i]), "\n")

# MLPRegressor
num_layers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
num_neurons = 20
mlp_train_errors = []
mlp_val_errors = []
mlp_test_errors = []

for i, num in enumerate(num_layers):
    train_errors = []
    val_errors = []
    test_errors = []

    for train_index, val_index in cv.split(y):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        hidden_layer_sizes = tuple([num_neurons] * num)

        mlp_regr = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,
max_iter=1000, random_state=42)
        mlp_regr.fit(X_train, y_train)

        y_pred_train = mlp_regr.predict(X_train)
        train_error = mean_squared_error(y_train, y_pred_train)
        train_errors.append(train_error)

        y_pred_val = mlp_regr.predict(X_val)
        val_error = mean_squared_error(y_val, y_pred_val)
        val_errors.append(val_error)

        y_pred_test = mlp_regr.predict(X_test)
        test_error = mean_squared_error(y_test, y_pred_test)
        test_errors.append(test_error)

    avg_train_error = math.log(sum(train_errors) / len(train_errors))
    avg_val_error = math.log(sum(val_errors) / len(val_errors))
    avg_test_error = math.log(sum(test_errors) / len(test_errors))
    mlp_train_errors.append(avg_train_error)
    mlp_val_errors.append(avg_val_error)
    mlp_test_errors.append(avg_test_error)

plt.figure(figsize=(10, 30))

plt.plot(num_layers, mlp_train_errors, label='Train')
plt.plot(num_layers, mlp_val_errors, label='Validation')
plt.plot(num_layers, mlp_test_errors, label='Test')
plt.xticks(num_layers)
plt.legend(loc='upper left')

plt.xlabel('Layers')
```

```
plt.ylabel('Loss')
plt.title('Train vs validation vs test loss')
plt.show()

# for i, layer in enumerate(num_layers):
#     print("MLPRegressor with ", layer, " layers has train error ",
math.log(mlp_train_errors[i]), "\n",
#           "MLPRegressor with ", layer, " layers has validation error ",
math.log(mlp_val_errors[i]), "\n",
#           "MLPRegressor with ", layer, " layers has test error ",
math.log(mlp_test_errors[i]), "\n\n")
```