Predicting the future price of the market index S&P 500 using Linear Regression and K Nearest Neighbour

CS-C3240 - Machine Learning

Introduction:

Computer science and machine learning has been a big part of the stock market trading for a few decades now and a great variety of mathematical indicators for stock trading has been developed over these past years. These indicators are very often based on the historical prices of the stocks or indexes which are being traded and the methods are used by many professionals. It is often said that the best results in stock trading comes when multiple different indicators are indicating the same thing, which is why machine learning and deep learning methods can be very useful to predict the stock market. More detailed information about indicators and theory of trading can be found for example in Investopedia (Folger 2022). This machine learning project is predicting the close price for the market index S&P 500. This could be useful information, when deciding whether to sell the stock or buy it on that given day.

Outline of this report consist of Introduction, Problem Solving, Methods, Results and Conclusions. Section "Problem Solving" defines what exactly this project is trying to do as well as it explains the origin of the dataset. In section "Methods" is explained how the problem is approached using the data set. In sections "Results" and "Conclusions" the results are presented and discussed.

Problem Formulation:

Even though the best possible result would be achieved by combining multiple different indicators for predicting the future close price S&P500 index, because of tight schedule of this course, this application will be using a very simple strategy. In addition to historical price data, also two extra features will be calculated and used in the prediction process: 1) repetitive peak points of the index (where the price movement has stopped and turned around, either up or down) and 2) price variability during each day (datapoint), which is simply a difference between highest and lowest price of the day. These values can be useful because, when price movement stops at a certain level multiple times it becomes so called resistance level and it is likely that movement will stop there in future as well. Usually near these resistance levels the variability in prices decreases.

Prediction is done by using supervised linear regression and supervised K nearest neighbour (kNN) regression methods. The features are the historical prices of S&P 500 index which are (continuous numerical), calculated peak points (binary) and variability of prices during each day (continuous numerical). The label is future close price for the index which is continuous numerical.

Methods:

The daily stock market data was downloaded from website WSJ Market (2022) as csv file. The data includes 1258 data points and each data point represent one day during 3rd of October 2017 – 3rd of October 2022. There are opening, close, highest, and lowest prices listed for each day. Opening and closing prices means the price in US dollars where the trading day started and ended, and the highest and lowest prices are the highest and lowest prices of the day.

Pre-processing of the data included formatting data type for dates, adding a column which has the difference between "High" and "Low", and finding the turning points, where the price has either bottomed or peaked and save this as binary data column (1 = yes, 0=no).

Linear regression will be used in this study because stock price data is continuous and numerical but also because over a long-term period the increasing price movement has been nearly linear. K nearest neighbour is another regression (also used as categorical) method which will be used as a comparison to linear regression.

Mean squared error is commonly used with linear regression, so it will be adapted in this project as well. Also, r2 score will be calculated to estimate the model. Both methods measure how far observed values differ from the average of predicted values.

The data is split roughly into 60% training, 20% validation and 20% test data randomly without shuffle. This is because the data set is not too big and as much data points as possible should be used for the validation and test sets. Ideally data would be split by day, using the oldest data for training and the latest data for validation and testing but because it is not possible to validate kNN method with this kind of datasets. The method needs neighbouring data around to make predictions and therefore the split needs to be random.

Results:

The loss function errors mean squared error and R2 are presented in the tables 1 and 2 below for both methods. R2 scores are very good for both methods, nearly perfect. This can be seen also in Figure 1, where the predicted close prices are plotted with the actual prices. Both graphs look similar.

However, there is some variance in mean squared errors. The validation data error relatively to training data is much larger for the kNN method than it is for the linear regression. This is a sign of overfitting. Even though the training data error is bigger for linear regression, the validation data error is much smaller than for kNN method. Therefore, linear regression is the final chosen method.

Table 1. Errors for linear regression

	Mean squared error	R2 Score
Training	217.422	0.999541
Validation	198.977	0.999574
Test	181.682	0.999604

Table 2. Errors for k Nearest Neighbour

	Mean squared error	R2 Score
Training	206.869	0.999563
Validation	463.925	0.999006

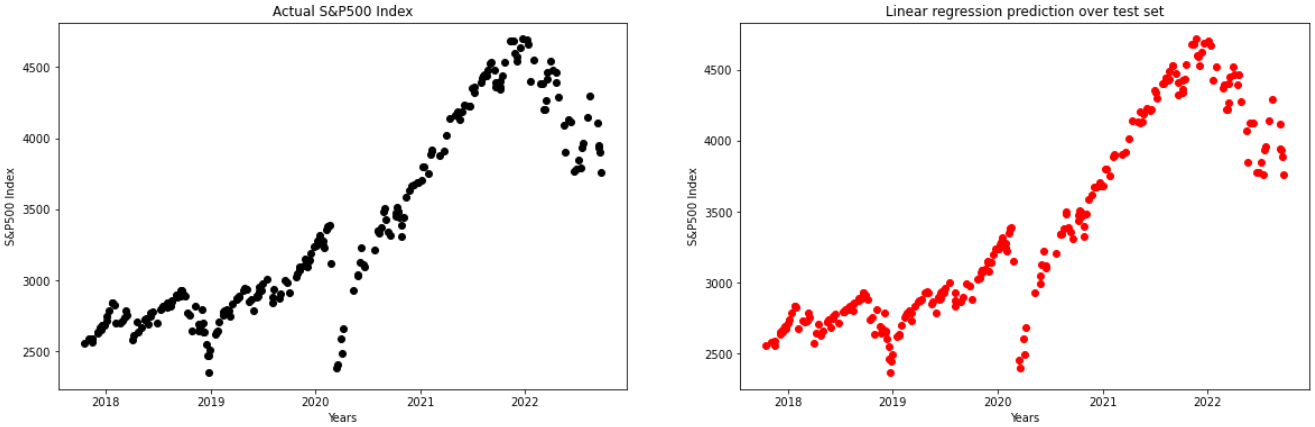

Fig. 1: Plotted test set data for linear regression, on the left the actual close prices of S&P 500 index and on the right the predicted prices.

The test error of the final chosen method is 181.6 USD which is quite big, if used in trading. In percentage it means 3.5-5% of the value of the whole index. Normally the index moves only 1-2% in a day, so the prediction is not very useful in real life. There are no signs of overfitting, since the validation error is little bit smaller than the training error.

Conclusion:

This method was found to be useless for real life trading and multiple problems were found. As mentioned in the previous chapter, the error margin is too big, and that is why it doesn't make sense to use the model in real trading. Another problem is that the model uses data such as highest and lowest price which can change during the day. Using historical data, which won't change, makes the model much better than it is. This is because the highest, lowest, and open prices are highly correlative with the close price, and if these values change dramatically during the trading day, also the close price changes dramatically and the error becomes bigger. Finally, third problem found was the historical data is always overrepresented and the most recent data is underrepresented. This is a problem because the interest is in the most recent data. This could be solved by balancing the data, but there's risks in it too because it is a difficult to say, what data should be highlighted and where to draw the line of "old" and "new" data.

Therefore, some other features should be used either in addition or on its own. These features need to be something that are available and not changing in time. These features could be for example fundamental factors which can be measured, such as number of news related to the index.

References:

Boxer S. 2019. S&P 500 Stock Price Prediction Using Machine Learning and Deep Learning. Medium. Viewed 04/10/2022: https://medium.com/shiyan-boxer/s-p-500-stock-price-prediction-using-machine-learning-and-deep-learning-328b1839d1b6

Folger J. 2022. Using Trading Indicators Effectively. Investopedia. Viewed 03/10/2022: https://www.investopedia.com/articles/trading/12/using-trading-indicators-effectively.asp

WSJ Markets, S&P 500 Index. 2022. Viewed in 03/10/2022:
https://www.wsj.com/market-data/quotes/index/SPX/historical-prices

Appendix

SP500_LinearRegression_kNN

October 5, 2022

1 Setup:

Importing modules, ensure MatplotLib plots figures inline and prepare a function to save the figures.

[ ]:
```
# Common imports
import numpy as np
import os
import pandas as pd
import sklearn
import datetime as dt

# For plotting
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt

#For preprocessing and modelling data
from sklearn import preprocessing
from sklearn import neighbors
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

#validating model
from sklearn.metrics import (mean_squared_error,
                             r2_score)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "end_to_end_project"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
```

```
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

2 Get the data from CSV

Reading the file and checking first 5 rows

[ ]:
```
df = pd.read_csv('SP500_WSJMarkets_Oct2020to2022.csv')
df.head(5)
```

2.0.1 Preprocessing data:

-Formatting column “Date”

[ ]:
```
#format colum Date from Sep 12, 2022 to 2022-09-12
df["Date"] = pd.to_datetime(df["Date"])
```

Two columns are added: -Difference between the highest and lowest prices indicating price variance during the day -Whether a day has been a turning point, this indicates possibility for resistance level, which is commonly used in trading

[ ]:
```
df2 = df.copy()
#difference between columns High and Low
df2["Hi_Low_Difference"] = (df2["High"] - df2["Low"])

#interphase for the final column of "Turning_Point"
df2['HigherThanDayBefore'] = df2['Close'] > df2['Close'].shift(periods=-1,␣
 ↪freq=None, axis=0)
df2['HigherThanDayAfter'] = df2['Close'] > df2['Close'].shift(periods=1,␣
 ↪freq=None, axis=0)


#comparing the values in two columns created before and checking that␣
 ↪everything looks good
df2['Turning_Point'] = np.where( df2['HigherThanDayBefore'] ==␣
 ↪df2['HigherThanDayAfter'] , 1, 0)
df2.head(5)

df2 = df2.drop(['HigherThanDayBefore', 'HigherThanDayAfter'],axis=1) #deleting␣
 ↪extra columns
```

[ ]:
```
df2.head(5)
```

Now we have a dataset df2 with extra columns “Hi_Low_Difference” and “Turning_Point”, which we use in the prediction process. In column Turning_Point value 1 means that the price was at its turning point on that day.

3 Viewing data

Next we'll see how the datapoints are distributed in columns Hi_Low_difference and Turning_Points.

[ ]:
```
fig, axes = plt.subplots(2, 1, figsize=(20,10)) # create a figure with two axes␣
 ↪(1 row,2 columns) on it

#filtering out only the rows, which have been turning points
TP_data = df2[df2['Turning_Point'] == 1]
TP_data = TP_data['Close']

axes[0].hist(TP_data, bins=30)
axes[0].set_title('Turning points',size=15)
axes[0].set_ylabel("# times prices has turned",size=15)
axes[0].set_xlabel("Close prices USD",size=15)

axes[1].scatter(df2['Close'],df2['Hi_Low_Difference'])
axes[1].set_title('Difference between highest and lowest price',size=15)
axes[1].set_ylabel("Difference of the highest and lowest price of the␣
 ↪day",size=15)
axes[1].set_xlabel("Close prices USD",size=15)


fig.tight_layout()
plt.show()
save_fig("plots")
```

From the plots we can see the majority of the turning points and the smallest variability of prices during a day are below 3000 USD, which makes sense because the data is from past five years and therefore lower prices will always be overrepresented in historical data. Because the price of the index is now at around 3600 USD, we are more interested in the values closer to that. Based on the historgram plot there is weak indication of resistance levels between 3000 - 3400 USD approximately and the scatter plot hghlights prices 3400 USD and 3200 USD approximately. These price levels will be taken into account in the section results.

3.1 Processing data for training

[ ]:
```
#creating a separate dataset
data = pd.DataFrame(index=range(0,len(df)),columns=['Date','Close', 'Open',␣
 ↪'High', 'Low', 'Turning_Point', 'Hi_Low_Difference'])

for i in range(0,len(data)):
    data['Date'][i] = df2['Date'][i]
    data['Close'][i] = df2['Close'][i]
    data['Open'][i] = df2['Open'][i]
    data['High'][i] = df2['High'][i]
    data['Low'][i] = df2['Low'][i]
```

```
    data['Turning_Point'][i] = df2['Turning_Point'][i]
    data['Hi_Low_Difference'][i] = df2['Hi_Low_Difference'][i]

#making sure that the data is sorted by date
data = data.sort_index(ascending=False, axis=0)

#setting numerical date as index for Linear Regression
data.index = data['Date'].map(dt.datetime.toordinal)
```

[ ]:
```
#split into train, validation and test datasets 60/20/20 %
from sklearn.model_selection import train_test_split

train, test = train_test_split(data, test_size=0.2, shuffle = True,␣
 ↪random_state = 42)

train, val = train_test_split(train, test_size=0.25, shuffle = True,␣
 ↪random_state = 42)
```

3.2 Linear Regression

3.2.1 Training the model and calculating errors

[ ]:
```
#deleting column Date for Linear Regression to avoid error
train_No_Date = train.drop(['Date'],axis=1)

#Training data
#creating features and target
X_train = train_No_Date.drop('Close', axis=1)
y_train = train_No_Date['Close']

#training the model and predicting using training data
model = LinearRegression()
model.fit(X_train, y_train)
predictions_train = model.predict(X_train)
```

[ ]:
```
#Validation data
#deleting column Date for Linear Regression
val_No_Date = val.drop(['Date'],axis=1)

#creating features and target
X_val = val_No_Date.drop('Close', axis=1)
y_val = val_No_Date['Close']

#predicting with validation data
predictions_val = model.predict(X_val)
```

[ ]:
```
#Test data

#deleting column Date for Linear Regression avoiding error
test_No_Date = test.drop(['Date'],axis=1)

#creating features and target
X_test = test_No_Date.drop('Close', axis=1)
y_test = test_No_Date['Close']

#predicting using test data
predictions_test = model.predict(X_test)
```

[ ]:
```
#Calculating errors
mse_train_lr = mean_squared_error(predictions_train, y_train)
r2_train_lr = r2_score(y_train, predictions_train)
mse_val_lr = mean_squared_error(predictions_val, y_val)
r2_val_lr = r2_score(y_val, predictions_val)
mse_test_lr = mean_squared_error(predictions_test, y_test)
r2_test_lr = r2_score(y_test, predictions_test)

from tabulate import tabulate
errors = [['Training', mse_train_lr, r2_train_lr],
['Validation', mse_val_lr, r2_val_lr],
['Test', mse_test_lr, r2_test_lr]]
print(tabulate(errors, headers=["Mean squared error", "R2 Score"]))
```

[ ]:
```
#Plotting results
#Training set
fig = plt.figure(figsize=(20,6))

for i in range(2):
    ax = fig.add_subplot(1, 2, i+1)
    if i == 0:
        plt.scatter(train['Date'], y_train, c='black')
        plt.title('Actual S&P500 Index')
    else:
        plt.scatter(train['Date'], predictions_train, c='red')
        plt.title('Linear regression prediction over training set')
    plt.xlabel('Years')
    plt.ylabel('S&P500 Index')

plt.show()

#Validation set
fig = plt.figure(figsize=(20,6))

for i in range(2):
```

```
    ax = fig.add_subplot(1, 2, i+1)
    if i == 0:
        plt.scatter(val['Date'], y_val, c='black')
        plt.title('Actual S&P500 Index')
    else:
        plt.scatter(val['Date'], predictions_val, c='red')
        plt.title('Linear regression prediction over validation set')
    plt.xlabel('Years')
    plt.ylabel('S&P500 Index')

plt.show()

#Test set
fig = plt.figure(figsize=(20,6))

for i in range(2):
    ax = fig.add_subplot(1, 2, i+1)
    if i == 0:
        plt.scatter(test['Date'], y_test, c='black')
        plt.title('Actual S&P500 Index')
    else:
        plt.scatter(test['Date'], predictions_test, c='red')
        plt.title('Linear regression prediction over test set')
    plt.xlabel('Years')
    plt.ylabel('S&P500 Index')

plt.show()
save_fig("predictions_LR")
```

3.3 K Nearest Neighbour

3.3.1 Training the model and calculating errors

[ ]:
```
#using gridsearch to find the best parameter
params = {'n_neighbors':[2,3,4,5,6,7,8,9]}
knn = neighbors.KNeighborsRegressor()
model_KNN = GridSearchCV(knn, params, cv=5)

#fit the model and make prediction using training data
model_KNN.fit(X_train,y_train)
predictions_train_knn = model_KNN.predict(X_train)

#print out the number of neighbors used in the model
model_KNN.best_params_

#predictions using validation data
predictions_val_knn = model_KNN.predict(X_val)
```

```
#predictions using test data
predictions_test_knn = model_KNN.predict(X_test)

#calculating errors
mse_train_knn = mean_squared_error(predictions_train_knn, y_train)
r2_train_knn = r2_score(y_train, predictions_train_knn)

mse_val_knn = mean_squared_error(predictions_val_knn, y_val)
r2_val_knn = r2_score(y_val, predictions_val_knn)

mse_test_knn = mean_squared_error(predictions_test_knn, y_test)
r2_test_knn = r2_score(y_test, predictions_test_knn)

errors = [['Training', mse_train_knn, r2_train_knn],
['Validation', mse_val_knn, r2_val_knn],
['Test', mse_test_knn, r2_test_knn]]
print(tabulate(errors, headers=["Mean squared error", "R2 Score"]))
```

[ ]:
```
#Plotting results
#Training data
fig = plt.figure(figsize=(20,6))

for i in range(2):
    ax = fig.add_subplot(1, 2, i+1)
    if i == 0:
        plt.scatter(train['Date'], y_train, c='black')
        plt.title('Actual S&P500 Index')
    else:
        plt.scatter(train['Date'], predictions_train_knn, c='red')
        plt.title('kNN Prediction over training set')
    plt.xlabel('Years')
    plt.ylabel('S&P500 Index')

plt.show()

#Validation data
fig = plt.figure(figsize=(20,6))

for i in range(2):
    ax = fig.add_subplot(1, 2, i+1)
    if i == 0:
        plt.scatter(val['Date'], y_val, c='black')
        plt.title('Actual S&P500 Index')
    else:
        plt.scatter(val['Date'], predictions_val_knn, c='red')
        plt.title('kNN Prediction over validation set')
```

```
    plt.xlabel('Years')
    plt.ylabel('S&P500 Index')

plt.show()

#Test data
fig = plt.figure(figsize=(20,6))

for i in range(2):
    ax = fig.add_subplot(1, 2, i+1)
    if i == 0:
        plt.scatter(test['Date'], y_test, c='black')
        plt.title('Actual S&P500 Index')
    else:
        plt.scatter(test['Date'], predictions_test_knn, c='red')
        plt.title('kNN Prediction over test set')
    plt.xlabel('Years')
    plt.ylabel('S&P500 Index')

plt.show()
save_fig("predictions_kNN")
```

[ ]: