S&P500 Prediction

February 12, 2020

[1]:
```
# Based on "Stock Market Forecasting Using Machine Learning Algorithms"
# by Shunrong Shen, Haomiao Jiang, Tongda Zhang
# https://pdfs.semanticscholar.org/b68e/8d2f4d2c709bb5919b82effcb6a7bbd3db37.
 ↪pdf
# Data from yahoo.com, macrotrends.net, investing.com (2001-01-01 to␣
 ↪2020-01-24)

# Get Data from SQL Server
import pandas as pd
import numpy as np
import pyodbc
import matplotlib.pyplot as plt
import sklearn.decomposition
import random
from sklearn.model_selection import train_test_split

NUM_DAYS = 30

connection = pyodbc.connect("Driver={SQL Server Native Client 11.0};"
                        "Server=DESKTOP-2JHG1EA\\SQLEXPRESS;"
                        "Database=Sandbox;"
                        "Trusted_Connection=yes;")
df = pd.read_sql(
    'SELECT * FROM [Sandbox].[dbo].[PredictionDataSetSp500] ORDER BY␣
 ↪[FeatureDate]',
    connection)

# Save some random periods of data for profit return % calculation
def get_random_period(df):
    row_count = df.shape[0]
    random_index = random.randint(1, row_count - NUM_DAYS)
    random_period = df[random_index: random_index + NUM_DAYS]
    before_period = df[0:random_index - 1]
    after_period = df[random_index + 1 + NUM_DAYS:row_count]
    df = before_period.append(after_period, ignore_index=True)
    return df, random_period
```

```
df, random_period_1 = get_random_period(df)
df, random_period_2 = get_random_period(df)
df, random_period_3 = get_random_period(df)
df, random_period_4 = get_random_period(df)
df, random_period_5 = get_random_period(df)

# Split data
def get_x_and_y(df):
    y = df[['Y_Index_GSPC']]
    y = np.where(df['Y_Index_GSPC'] > 0, 1, 0)
    X = df[['X_Index_SSMI', 'X_Index_N225', 'X_Index_AXJO', 'X_Index_HSI',␣
 ↪'X_Index_N100', 'X_Index_FTSE', 'X_Index_GDAXI']]
    return X, y

X, y = get_x_and_y(df)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[2]:
```
# Use AutoML
from tpot import TPOTClassifier

model = TPOTClassifier(verbosity=2, n_jobs=-1, config_dict='TPOT light')
model.fit(X_train, y_train)
```

```
HBox(children=(IntProgress(value=0, description='Optimization Progress', max=10100, style=Prog
```

```
Generation 1 - Current best internal CV score: 0.7813377618659516
Generation 2 - Current best internal CV score: 0.7830977042362225
Generation 3 - Current best internal CV score: 0.7830977042362225
Generation 4 - Current best internal CV score: 0.7830977042362225
Generation 5 - Current best internal CV score: 0.7830977042362225
Generation 6 - Current best internal CV score: 0.7841565197559983
Generation 7 - Current best internal CV score: 0.7841565197559983
Generation 8 - Current best internal CV score: 0.7841565197559983
Generation 9 - Current best internal CV score: 0.7841565197559983
Generation 10 - Current best internal CV score: 0.7859133723558998
Generation 11 - Current best internal CV score: 0.7859133723558998
Generation 12 - Current best internal CV score: 0.7859133723558998
Generation 13 - Current best internal CV score: 0.7859133723558998
Generation 14 - Current best internal CV score: 0.7859133723558998
Generation 15 - Current best internal CV score: 0.7859133723558998
Generation 16 - Current best internal CV score: 0.7866188353624098
Generation 17 - Current best internal CV score: 0.7866188353624098
Generation 18 - Current best internal CV score: 0.7866188353624098
Generation 19 - Current best internal CV score: 0.7866188353624098
Generation 20 - Current best internal CV score: 0.7866188353624098
Generation 21 - Current best internal CV score: 0.7866188353624098
```

```
Generation 22 - Current best internal CV score: 0.7866188353624098
Generation 23 - Current best internal CV score: 0.7866194651036684
Generation 24 - Current best internal CV score: 0.7866194651036684
Generation 25 - Current best internal CV score: 0.7866194651036684
Generation 26 - Current best internal CV score: 0.7869697103840687
Generation 27 - Current best internal CV score: 0.7869697103840687
Generation 28 - Current best internal CV score: 0.7869697103840687
Generation 29 - Current best internal CV score: 0.7869697103840687
Generation 30 - Current best internal CV score: 0.7869697103840687
Generation 31 - Current best internal CV score: 0.7869715690484769
Generation 32 - Current best internal CV score: 0.7869715690484769
Generation 33 - Current best internal CV score: 0.7869715690484769
Generation 34 - Current best internal CV score: 0.7869715690484769
Generation 35 - Current best internal CV score: 0.7887290339271412
Generation 36 - Current best internal CV score: 0.7887290339271412
Generation 37 - Current best internal CV score: 0.7887290339271412
Generation 38 - Current best internal CV score: 0.7887290339271412
Generation 39 - Current best internal CV score: 0.7887290339271412
Generation 40 - Current best internal CV score: 0.7887290339271412
Generation 41 - Current best internal CV score: 0.7887290339271412
Generation 42 - Current best internal CV score: 0.7887290339271412
Generation 43 - Current best internal CV score: 0.7887290339271412
Generation 44 - Current best internal CV score: 0.7890805277759988
Generation 45 - Current best internal CV score: 0.7890805277759988
Generation 46 - Current best internal CV score: 0.7890805277759988
Generation 47 - Current best internal CV score: 0.7890805277759988
Generation 48 - Current best internal CV score: 0.7890805277759988
Generation 49 - Current best internal CV score: 0.7890805277759988
Generation 50 - Current best internal CV score: 0.7890805277759988
Generation 51 - Current best internal CV score: 0.7890805277759988
Generation 52 - Current best internal CV score: 0.7890805277759988
Generation 53 - Current best internal CV score: 0.7890805277759988
Generation 54 - Current best internal CV score: 0.7890805277759988
Generation 55 - Current best internal CV score: 0.7890805277759988
Generation 56 - Current best internal CV score: 0.7890805277759988
Generation 57 - Current best internal CV score: 0.7890805277759988
Generation 58 - Current best internal CV score: 0.7890805277759988
Generation 59 - Current best internal CV score: 0.7890805277759988
Generation 60 - Current best internal CV score: 0.7890805277759988
Generation 61 - Current best internal CV score: 0.7890811466031974
Generation 62 - Current best internal CV score: 0.7890811466031974
Generation 63 - Current best internal CV score: 0.7890811466031974
Generation 64 - Current best internal CV score: 0.7890811466031974
Generation 65 - Current best internal CV score: 0.7890811466031974
Generation 66 - Current best internal CV score: 0.7890811466031974
Generation 67 - Current best internal CV score: 0.7890811466031974
Generation 68 - Current best internal CV score: 0.7890811466031974
Generation 69 - Current best internal CV score: 0.7890811466031974
```

```
Generation 70 - Current best internal CV score: 0.7890811466031974
Generation 71 - Current best internal CV score: 0.7890811466031974
Generation 72 - Current best internal CV score: 0.7890811466031974
Generation 73 - Current best internal CV score: 0.7890811466031974
Generation 74 - Current best internal CV score: 0.7890811466031974
Generation 75 - Current best internal CV score: 0.7890811466031974
Generation 76 - Current best internal CV score: 0.7890811466031974
Generation 77 - Current best internal CV score: 0.7894375976180805
Generation 78 - Current best internal CV score: 0.7894375976180805
Generation 79 - Current best internal CV score: 0.7894375976180805
Generation 80 - Current best internal CV score: 0.7894375976180805
Generation 81 - Current best internal CV score: 0.7894375976180805
Generation 82 - Current best internal CV score: 0.7894375976180805
Generation 83 - Current best internal CV score: 0.7897903313041476
Generation 84 - Current best internal CV score: 0.7897903313041476
Generation 85 - Current best internal CV score: 0.7897903313041476
Generation 86 - Current best internal CV score: 0.7897903313041476
Generation 87 - Current best internal CV score: 0.7897903313041476
Generation 88 - Current best internal CV score: 0.7897903313041476
Generation 89 - Current best internal CV score: 0.7897903313041476
Generation 90 - Current best internal CV score: 0.7897903313041476
Generation 91 - Current best internal CV score: 0.7897903313041476
Generation 92 - Current best internal CV score: 0.7897903313041476
Generation 93 - Current best internal CV score: 0.7897903313041476
Generation 94 - Current best internal CV score: 0.7897903313041476
Generation 95 - Current best internal CV score: 0.7897903313041476
Generation 96 - Current best internal CV score: 0.7897903313041476
Generation 97 - Current best internal CV score: 0.7897903313041476
Generation 98 - Current best internal CV score: 0.7897903313041476
Generation 99 - Current best internal CV score: 0.7897903313041476
Generation 100 - Current best internal CV score: 0.7897903313041476

Best pipeline: KNeighborsClassifier(MaxAbsScaler(CombineDFs(input_matrix,
RBFSampler(MaxAbsScaler(SelectFwe(StandardScaler(input_matrix), alpha=0.031)),
gamma=0.30000000000000004))), n_neighbors=99, p=2, weights=uniform)
```

[2]:
```
TPOTClassifier(config_dict='TPOT light', crossover_rate=0.1, cv=5,
               disable_update_check=False, early_stop=None, generations=100,
               max_eval_time_mins=5, max_time_mins=None, memory=None,
               mutation_rate=0.9, n_jobs=-1, offspring_size=None,
               periodic_checkpoint_folder=None, population_size=100,
               random_state=None, scoring=None, subsample=1.0, template=None,
               use_dask=False, verbosity=2, warm_start=False)
```

[3]:
```
print('Score', model.score(X_test, y_test))
print('')

# Calculate Profit Return
```

```
def print_profit_return(model, random_period, period):
    X_random_period, y_random_period = get_x_and_y(random_period)
    predictions = model.predict(X_random_period)
    capital_initial = 10000
    capital_actual = capital_initial
    capital_model = capital_initial
    for index, prediction in enumerate(predictions):
        percent_change = random_period.iloc[index]['Y_Index_GSPC']
        capital_actual += capital_actual * percent_change
        if(prediction == 1):
            capital_model += capital_model * percent_change

    thirty_day_return_actual = round(((capital_actual/capital_initial) - 1) *␣
↪100, 2)
    thirty_day_return_model = round(((capital_model/capital_initial) - 1) *␣
↪100, 2)

    print(period, 'Results:')
    print('Actual Capital: $' + str(round(capital_actual, 2)))
    print('Actual Returns Monthly: ' + str(thirty_day_return_actual) + '%')
    print('Model Capital: $' + str(round(capital_model, 2)))
    print('Model Returns Monthly: ' + str(thirty_day_return_model) + '%')
    print('')

print_profit_return(model, random_period_1, 'Period 1')
print_profit_return(model, random_period_2, 'Period 2')
print_profit_return(model, random_period_3, 'Period 3')
print_profit_return(model, random_period_4, 'Period 4')
print_profit_return(model, random_period_5, 'Period 5')
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\preprocessing\_function_transformer.py:97: FutureWarning: The
default validate=True will be replaced by validate=False in 0.22.
  "validate=False in 0.22.", FutureWarning)
```

```
Score 0.7988748241912799
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\preprocessing\_function_transformer.py:97: FutureWarning: The
default validate=True will be replaced by validate=False in 0.22.
  "validate=False in 0.22.", FutureWarning)
```

```
Period 1 Results:
Actual Capital: $10607.77
Actual Returns Monthly: 6.08%
Model Capital: $10100.0
Model Returns Monthly: 1.0%
```

```
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\preprocessing\_function_transformer.py:97: FutureWarning: The
default validate=True will be replaced by validate=False in 0.22.
  "validate=False in 0.22.", FutureWarning)

Period 2 Results:
Actual Capital: $10293.74
Actual Returns Monthly: 2.94%
Model Capital: $10614.16
Model Returns Monthly: 6.14%


C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\preprocessing\_function_transformer.py:97: FutureWarning: The
default validate=True will be replaced by validate=False in 0.22.
  "validate=False in 0.22.", FutureWarning)

Period 3 Results:
Actual Capital: $10193.88
Actual Returns Monthly: 1.94%
Model Capital: $10201.0
Model Returns Monthly: 2.01%


C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\preprocessing\_function_transformer.py:97: FutureWarning: The
default validate=True will be replaced by validate=False in 0.22.
  "validate=False in 0.22.", FutureWarning)

Period 4 Results:
Actual Capital: $10194.88
Actual Returns Monthly: 1.95%
Model Capital: $10201.0
Model Returns Monthly: 2.01%


C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\preprocessing\_function_transformer.py:97: FutureWarning: The
default validate=True will be replaced by validate=False in 0.22.
  "validate=False in 0.22.", FutureWarning)

Period 5 Results:
Actual Capital: $9698.16
Actual Returns Monthly: -3.02%
Model Capital: $10100.0
Model Returns Monthly: 1.0%
```

[ ]: