Assignment4_Dhyey_Mavani_Prog_For_Quant_and_Comp_Fin

April 30, 2023

[1]:
```
# importing the necessary packages
import yfinance as yf
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

1 Question 1: Part (a)

[2]:
```
# intializing tickers list
tickers_list = ["AAPL", "HD", "JNJ", "JPM", "MSFT", "UNH", "V", "XOM"]
# intitializing start and end dates
start_date, end_date = '2019-01-02', '2021-01-01'
# downloading the data from yahoo finance
data = yf.download(tickers_list, start = start_date, end = end_date)
```

```
[*********************100%***********************]  8 of 8 completed
```

[3]:
```
# getting the data for "Adj Close" prices for all tickers of interest
tickers_prices = data["Adj Close"]
tickers_prices
```

[3]:
```
                  AAPL          HD         JNJ         JPM        MSFT  \
Date
2019-01-02   38.047043  155.546707  114.144203   86.554695   96.632660
2019-01-03   34.257275  152.118347  112.330399   85.324593   93.077728
2019-01-04   35.719696  156.638351  114.215675   88.470139   97.406700
2019-01-07   35.640198  159.723816  113.483032   88.531631   97.530937
2019-01-08   36.319607  160.490692  116.118828   88.364693   98.238121
…                  …           …           …           …           …
2020-12-24  130.205795  257.002350  143.879669  115.979340  218.300949
2020-12-28  134.862656  255.418091  144.559128  116.743095  220.466812
2020-12-29  133.067001  252.515335  145.455582  116.435738  219.672974
2020-12-30  131.932373  251.633102  147.257980  116.761726  217.252304
2020-12-31  130.916168  251.974640  148.513046  118.354439  217.977539
```

```
                   UNH           V        XOM
Date
2019-01-02  228.546371  129.230560  55.378773
2019-01-03  222.313919  124.573524  54.528496
2019-01-04  224.913864  129.940323  56.538948
2019-01-07  225.345673  132.283432  56.832973
2019-01-08  228.358658  133.002853  57.246193
…                  …           …          …
2020-12-24  330.637726  205.466110  37.406952
2020-12-28  335.643982  209.335251  37.532845
2020-12-29  337.002319  211.048264  37.110214
2020-12-30  334.712585  214.976456  37.406952
2020-12-31  340.233063  215.340729  37.065266

[505 rows x 8 columns]
```

2 Question 1: Part (b)

[4]:
```python
# filling in the missing values using the method described
tickers_prices.fillna(method = "ffill", inplace = True)
```

[5]:
```python
returns = np.log(tickers_prices) - np.log(tickers_prices.shift(1))
returns = returns.dropna().loc["2019-01-04":"2020-12-30"]
```

[6]:
```python
dji_prices = yf.download("^DJI", start = start_date, end = end_date)["Adj␣
↪Close"]
dji_prices
```

```
[*********************100%***********************]  1 of 1 completed
```

[6]:
```
Date
2019-01-02    23346.240234
2019-01-03    22686.220703
2019-01-04    23433.160156
2019-01-07    23531.349609
2019-01-08    23787.449219
                  …
2020-12-24    30199.869141
2020-12-28    30403.970703
2020-12-29    30335.669922
2020-12-30    30409.560547
2020-12-31    30606.480469
Name: Adj Close, Length: 505, dtype: float64
```

[7]:
```python
dji_returns = np.log(dji_prices) - np.log(dji_prices.shift(1))
dji_returns = dji_returns.dropna()
```

```
dji_returns = dji_returns.loc["2019-01-07":]
```

[8]:
```
labels = dji_returns.apply(lambda x: 1 if x > 0 else -1)
```

[9]:
```
from sklearn.model_selection import train_test_split
returns_train, returns_test, labels_train, labels_test =␣
↪train_test_split(returns, labels, test_size=0.20, shuffle=False)
```

[10]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(returns_train)
scaled_returns_train = scaler.transform(returns_train)
scaled_returns_test = scaler.transform(returns_test)
```

3 Question 1: Part (c)

[11]:
```
from sklearn import svm
svm_model = svm.SVC(gamma = 1,C = 1,kernel = "rbf")
svm_model.fit(scaled_returns_train, labels_train)
```

[11]:
```
SVC(C=1, gamma=1)
```

[12]:
```
svm_model.score(scaled_returns_test, labels_test)
```

[12]:
```
0.594059405940594
```

4 Question 1: Part (d)

[13]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import TimeSeriesSplit

cv_scores = cross_val_score(svm_model,
                            scaled_returns_train,
                            labels_train,
                            cv = TimeSeriesSplit(n_splits=5))
```

[14]:
```
cv_scores.mean()
```

[14]:
```
0.5727272727272726
```

[15]:
```
cv_scores
```

[15]:
```
array([0.65151515, 0.65151515, 0.56060606, 0.46969697, 0.53030303])
```

5 Question 1: Part (e)

[16]:
```
from sklearn.model_selection import GridSearchCV
parameters_grid = [{'C': [0.1, 1.0, 10, 100],
                    'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 1, 5, 10]}]
grid = GridSearchCV(svm_model,
                    param_grid = parameters_grid,
                    cv = TimeSeriesSplit(n_splits=5),
                    return_train_score=True)
grid.fit(scaled_returns_train, labels_train)
best_model = grid.best_estimator_
print("Best Parameters from Grid Search are: ", grid.best_params_)
print("Best score from grid search is: ", grid.best_score_)
```

```
Best Parameters from Grid Search are:  {'C': 1.0, 'gamma': 1}
Best score from grid search is:  0.5727272727272726
```

6 Question 1: Part (f)

[17]:
```
dji_prices.loc["2020-09-15":"2020-10-05"]
```

[17]:
```
Date
2020-09-15    27995.599609
2020-09-16    28032.380859
2020-09-17    27901.980469
2020-09-18    27657.419922
2020-09-21    27147.699219
2020-09-22    27288.179688
2020-09-23    26763.130859
2020-09-24    26815.439453
2020-09-25    27173.960938
2020-09-28    27584.060547
2020-09-29    27452.660156
2020-09-30    27781.699219
2020-10-01    27816.900391
2020-10-02    27682.810547
2020-10-05    28148.640625
Name: Adj Close, dtype: float64
```

[18]:
```
dji_prices[441] == dji_prices.loc["2020-10-01"]
```

[18]:
```
True
```

[19]:
```
# Simulate a trading strategy using the previously trained SVM model with Gamma␣
↪= C = 1
```

```
capital = 10000  # initial capital amount
shares = 0  # number of shares held

returns_test_after_oct_1 = returns_test["2020-10-01":]
predicted_labels = best_model.predict(scaler.
 ↪transform(returns_test_after_oct_1))

for i in range(len(predicted_labels)):
    j = 441 + i # starting 1st October
    signal = predicted_labels[i]
    if signal == 1 and shares == 0:
        shares = capital / dji_prices[j]
        capital = 0
    elif signal == -1 and shares > 0:
        capital = shares * dji_prices[j]
        shares = 0

# Compute the final amount at market close of Dec 31st 2020
final_amount = (capital + shares * dji_prices[-1])
print(f'Final amount for model based trading strategy: ${final_amount:.5f}')
```

```
Final amount for model based trading strategy: $11880.39472
```

7 Question 1: Part (g)

[20]:
```
# Simple Buy and Hold Strategy Implementation

# Number of shares bought on Oct 1st 2020 with initial capital
num_shares = 10000 / dji_prices.loc["2020-10-01"]

# Final amount at market close of Dec 31st 2020
final_amount = num_shares * dji_prices.loc["2020-12-31"]

print(f"Final amount for buy-and-hold strategy: ${np.round(final_amount, 5)}")
```

```
Final amount for buy-and-hold strategy: $11002.83642
```

Compared to the buy and hold baseline strategy, our trading strategy based on predictions in the previous part made around 880 more dollars in profit. This tells us that our prediction is not completely useless, but also it is not very good. If we are considering deployment of this strategy, then I think there might not be much differences in profit among the above-mentioned strategies because of factors like imperfect execution and transactional costs.

[ ]:
```

```