Shea-FA692-HW5

April 19, 2023

1 FA692 Homework 5

2 Due: Wednesday, April 19 @ 11:59PM

Name: Ryan Shea

Date: 2023-04-18

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Set seed of random number generator
CWID = 10445281 #Place here your Campus wide ID number, this will personalize
#your results, but still maintain the reproduceable nature of using seeds.
#If you ever need to reset the seed in this assignment, use this as your seed
#Papers that use -1 as this CWID variable will earn 0's so make sure you change
#this value before you submit your work.
personal = CWID % 10000
np.random.seed(personal)
```

2.1 Question 1 (10pt)

2.1.1 Question 1.1

Use the `yfinance` package (or other method of your choice) to obtain the daily adjusted close prices for the S&P500 (`SPY`) from January 1, 2023 to March 15, 2023. You should inspect the dates for your data to make sure you are including everything appropriately. Create a data frame (or array) of the daily log returns of this stock; you may concatenate this to your price data. Use the `print` command to display your data.

[ ]:
```
import yfinance as yf

data = yf.download('SPY', start='2023-01-01', end='2023-03-15')
ret = data['Adj Close'].apply(np.log).diff().dropna()

print(ret.head())
```

```
[*********************100%***********************]  1 of 1 completed
Date
2023-01-04    0.007691
2023-01-05   -0.011479
2023-01-06    0.022673
2023-01-09   -0.000567
2023-01-10    0.006988
Name: Adj Close, dtype: float64

Date
2023-01-04    0.007691
2023-01-05   -0.011479
2023-01-06    0.022673
2023-01-09   -0.000567
2023-01-10    0.006988
Name: Adj Close, dtype: float64
```

2.1.2 Question 1.2

Scrape data from the Bloomberg `@business` Twitter account from January 1, 2023 to March 15, 2023. Save this data to a Data Frame with time stamps. Additionally, save all the collected data to a text file with time stamps. You will need to submit the text file along with your work (-5 points if not submitted).

Note: Bloomberg tweets sometimes include the pipe "|". I recomment using tilde "~" as a delimiter instead.

Hint: Because saving the tweets can take a long time, you can comment that code out before exporting to pdf.

```
[ ]: # import snscrape.modules.twitter as tw

# f = open('business.txt', 'w', encoding='utf-8')

# for tweet in tw.TwitterSearchScraper(query="(from:business) since:2023-01-01␣
 ↪until:2023-03-15").get_items():
#     date_str = tweet.date.strftime("%Y-%m-%d %H:%M:%S%z")
#     date_str = date_str[:-2] + ":" + date_str[-2:]
#     #f.write(date_str + "|" + tweet.content + "\n")
#     f.write(date_str + "~" + tweet.rawContent + "\n")
# f.close()

from datetime import datetime as dt
import pytz

business = []
dates = []
f = open('business.txt', 'r', encoding='utf-8')
```

```python
for l in f:
    line = l.split('~')
    date_str = line[0]
    try:
        date_time = dt.fromisoformat(date_str)
        date_time = date_time.astimezone(pytz.timezone("US/Eastern"))
        line[0] = date_time
        line[1] = line[1][:-1]
        business.append(line)
        dates.append(date_time.date())
    except:
        business[-1][1] += " "+l[:-1]
f.close()

business = pd.DataFrame(business, columns=['Time', 'Tweet'])
business['Date'] = dates

business
```

[ ]:

```
                           Time  \
0     2023-03-14 19:40:29-04:00
1     2023-03-14 19:40:29-04:00
2     2023-03-14 19:35:41-04:00
3     2023-03-14 19:31:07-04:00
4     2023-03-14 19:25:09-04:00
…                           …
26809 2022-12-31 19:00:09-05:00
26810 2022-12-31 19:00:09-05:00
26811 2022-12-31 19:00:09-05:00
26812 2022-12-31 19:00:09-05:00
26813 2022-12-31 19:00:08-05:00

                                                   Tweet        Date
0      One Japanese fintech firm is making it compuls…  2023-03-14
1      An unlikely startup guru has emerged in Japan,…  2023-03-14
2      Some US cities are late in making financial di…  2023-03-14
3      The shipping industry is looking to rethink ev…  2023-03-14
4      A biotech wants to cut fashion waste by using …  2023-03-14
…                                                  …         …
26809  Toymakers have found a new group of customers:…  2022-12-31
26810  Belarusian hackers and dissidents determined t…  2022-12-31
26811  Landlords are taking out millions in loans to …  2022-12-31
26812  It took a pandemic to make a dent in US inequa…  2022-12-31
26813  A planned train line in Mexico is billions ove…  2022-12-31

[26814 rows x 3 columns]
```

2.2 Question 2 (30pt)

2.2.1 Question 2.1

Use Latent Dirichlet Allocation to cluster the tweets into at least 8 topics. You may use your favorite method for tokenizing and vectorizing. Display the top 10 words for each of your topic. Add a column to your Data Frame of Tweets to label the topic associated with each Tweet.

[ ]:
```
# Pre-process for Latent Dirichlet Allocation
from nltk.tokenize import RegexpTokenizer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

# Initialize regex tokenizer
tokenizer = RegexpTokenizer(r'\w+') # Tokenizes by word

# Term frequency-inverse document frequency
tfidf =␣
↪TfidfVectorizer(lowercase=True,stop_words='english',ngram_range=(1,1),tokenizer=tokenizer.
↪tokenize)
tfidf_data = tfidf.fit_transform(business['Tweet'].tolist())
tfidf_feature_names = tfidf.get_feature_names_out()
```

[ ]:
```
# Run Latent Dirichlet Allocation
from sklearn.decomposition import LatentDirichletAllocation

K = 10 # Number of topics

# Try with term frequency-inverse document frequency:
LDA_TFIDF = LatentDirichletAllocation(n_components=K)
LDA_TFIDF_Matrix = LDA_TFIDF.fit_transform(tfidf_data) # Fits and transforms␣
↪the dataset
LDA_TFIDF_Components = LDA_TFIDF.components_ # Get the components

def display_topics(components, feature_names, no_top_words):
    for index, topic in enumerate(components):
        top_terms_list = [feature_names[i] for i in topic.argsort()[:
↪-no_top_words-1:-1]]
        print("Topic "+str(index+1)+": ",top_terms_list)

no_top_words = 10

display_topics(LDA_TFIDF_Components, tfidf_feature_names, no_top_words)
```

```
Topic 1:  ['t', 'https', 's', 'ukraine', 'president', 'biden', 'latest',
'russia', 'says', 'updates']
Topic 2:  ['t', 'https', 's', 'year', 'musk', 'new', 'elon', 'says',
'inflation', 'market']
Topic 3:  ['t', 'https', 's', 'new', 'opinion', 'billion', 'world', 'big',
'climate', 'people']
```

```
Topic 4:  ['t', 'https', 's', 'new', 'year', 'says', 'bank', 'crypto', 'world',
'billion']
Topic 5:  ['t', 'https', 's', 'new', 'opinion', 'year', 'says', 'york', 'world',
'london']
Topic 6:  ['t', 'https', 's', 'new', 'year', 'best', 'climate', 'tv', 'million',
'billion']
Topic 7:  ['s', 't', 'https', 'year', 'adani', 'new', 'china', 'market', 'says',
'know']
Topic 8:  ['t', 'https', 's', 'bank', 'rate', 'inflation', 'fed', 'says', 'new',
'year']
Topic 9:  ['t', 'https', 's', 'china', 'says', 'world', 'secretary', 'bank',
'week', 'year']
Topic 10:  ['t', 'https', 's', 'new', 'china', 'year', 'uk', 'says', 'people',
'finance']
```

[ ]:
```python
labels = np.argmax(LDA_TFIDF_Matrix, axis=1)
business['Cluster'] = labels
business
```

[ ]:
```
                            Time  \
0      2023-03-14 19:40:29-04:00
1      2023-03-14 19:40:29-04:00
2      2023-03-14 19:35:41-04:00
3      2023-03-14 19:31:07-04:00
4      2023-03-14 19:25:09-04:00
…                            …
26809 2022-12-31 19:00:09-05:00
26810 2022-12-31 19:00:09-05:00
26811 2022-12-31 19:00:09-05:00
26812 2022-12-31 19:00:09-05:00
26813 2022-12-31 19:00:08-05:00

                                                   Tweet        Date  Cluster
0      One Japanese fintech firm is making it compuls…  2023-03-14        9
1      An unlikely startup guru has emerged in Japan,…  2023-03-14        7
2      Some US cities are late in making financial di…  2023-03-14        6
3      The shipping industry is looking to rethink ev…  2023-03-14        6
4      A biotech wants to cut fashion waste by using …  2023-03-14        2
…                                                  …         …      …
26809  Toymakers have found a new group of customers:…  2022-12-31        2
26810  Belarusian hackers and dissidents determined t…  2022-12-31        3
26811  Landlords are taking out millions in loans to …  2022-12-31        5
26812  It took a pandemic to make a dent in US inequa…  2022-12-31        9
26813  A planned train line in Mexico is billions ove…  2022-12-31        0

[26814 rows x 4 columns]
```

2.2.2 Question 2.2

Using your favorite sentiment analyzer (e.g., `vaderSentiment`), find the average sentiment for each topic from your topic modeling for the headlines on each date that data was collected. Concatenate this sentiment score to your data frame of log returns. Use the `print` command to display your data.

[ ]:
```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
sentiment = []
analyzer = SentimentIntensityAnalyzer()
for tweet in business['Tweet']:
    vs = analyzer.polarity_scores(tweet)
    sentiment.append(vs['compound'])

business['Sentiment'] = sentiment

df = pd.DataFrame({'log': ret})
df['sent'] = business.pivot_table(index='Date', values='Sentiment',␣
 ↪aggfunc='mean')
print(df.head())
```

```
                 log      sent
Date
2023-01-04  0.007691 -0.010883
2023-01-05 -0.011479 -0.032084
2023-01-06  0.022673 -0.001990
2023-01-09 -0.000567  0.048762
2023-01-10  0.006988 -0.051226
```

2.3 Question 3 (20pt)

2.3.1 Question 3.1

Linearly regress SPY returns as a function of the lagged returns (2 lags). This should be of the form $r_t = \beta_0 + \beta_1 r_{t-1} + \beta_2 r_{t-2}$. Evaluate the performance of this model with the mean squared error of the training data.

[ ]:
```
for i in range(1, 3):
    df[f'log_{i}'] = df['log'].shift(i)
    df[f'sent_{i}'] = df['sent'].shift(i)



# average sentiment on each day for each topic
for k in range(K):
    df[f'cluster_{k}'] = business[business['Cluster'] == k] \
        .pivot_table(index='Date', values='Sentiment', aggfunc='mean')
    # lagged sentiment for each topic
    for i in range(1, 3):
        df[f'cluster_{k}_{i}'] = df[f'cluster_{k}'].shift(i)
```

```
df = df.dropna()

print(f"{df.shape = }")
df.head()
```

```
df.shape = (46, 36)
```

[ ]:

```
                 log      sent     log_1    sent_1     log_2    sent_2  \
Date
2023-01-06  0.022673 -0.001990 -0.011479 -0.032084  0.007691 -0.010883
2023-01-09 -0.000567  0.048762  0.022673 -0.001990 -0.011479 -0.032084
2023-01-10  0.006988 -0.051226 -0.000567  0.048762  0.022673 -0.001990
2023-01-11  0.012569 -0.006794  0.006988 -0.051226 -0.000567  0.048762
2023-01-12  0.003634  0.012535  0.012569 -0.006794  0.006988 -0.051226

            cluster_0  cluster_0_1  cluster_0_2  cluster_1  …  cluster_6_2  \
Date                                                        …
2023-01-06  -0.023586    -0.019584    -0.003480  -0.010891  …     0.095122
2023-01-09   0.017994    -0.023586    -0.019584   0.031297  …     0.040583
2023-01-10  -0.098214     0.017994    -0.023586   0.123016  …    -0.002362
2023-01-11  -0.041914    -0.098214     0.017994  -0.004346  …     0.081896
2023-01-12   0.055011    -0.041914    -0.098214  -0.059940  …    -0.133607

            cluster_7  cluster_7_1  cluster_7_2  cluster_8  cluster_8_1  \
Date
2023-01-06   0.009463    -0.119121    -0.018074  -0.023271    -0.087319
2023-01-09   0.078900     0.009463    -0.119121   0.067354    -0.023271
2023-01-10  -0.005892     0.078900     0.009463   0.000242     0.067354
2023-01-11  -0.030783    -0.005892     0.078900   0.026706     0.000242
2023-01-12   0.066988    -0.030783    -0.005892  -0.063157     0.026706

            cluster_8_2  cluster_9  cluster_9_1  cluster_9_2
Date
2023-01-06     0.058240  -0.017962    -0.037485    -0.062720
2023-01-09    -0.087319   0.111824    -0.017962    -0.037485
2023-01-10    -0.023271   0.037836     0.111824    -0.017962
2023-01-11     0.067354  -0.010175     0.037836     0.111824
2023-01-12     0.000242   0.091786    -0.010175     0.037836

[5 rows x 36 columns]
```

[ ]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

def linreg(X):
```

```
    y = df['log']

    lr = LinearRegression().fit(X, y)
    y_pred = lr.predict(X)

    print(f"    Intercept: {lr.intercept_}", end='\n\n')

    for feat, coef in zip(X.columns, lr.coef_):
        print(f"{feat:>13}: {coef}")

    MSE = mean_squared_error(y, y_pred)
    print()
    print(f"{MSE = }")

    return lr, MSE

X = df[['log_1', 'log_2']]

lr1, mse1 = linreg(X)
```

```
    Intercept: 0.0007263457920817806

        log_1: 0.05614851635772608
        log_2: -0.12133150745716237

MSE = 0.00011083584269359552
```

[ ]:
```
df.columns
```

[ ]:
```
Index(['log', 'sent', 'log_1', 'sent_1', 'log_2', 'sent_2', 'cluster_0',
       'cluster_0_1', 'cluster_0_2', 'cluster_1', 'cluster_1_1', 'cluster_1_2',
       'cluster_2', 'cluster_2_1', 'cluster_2_2', 'cluster_3', 'cluster_3_1',
       'cluster_3_2', 'cluster_4', 'cluster_4_1', 'cluster_4_2', 'cluster_5',
       'cluster_5_1', 'cluster_5_2', 'cluster_6', 'cluster_6_1', 'cluster_6_2',
       'cluster_7', 'cluster_7_1', 'cluster_7_2', 'cluster_8', 'cluster_8_1',
       'cluster_8_2', 'cluster_9', 'cluster_9_1', 'cluster_9_2'],
      dtype='object')
```

[ ]:
```
X_all = []  # all lagged variables
for c in df.columns:
    if c.endswith('_1') or c.endswith('_2'):
        if c not in ['cluster_1', 'cluster_2']:
            X_all.append(c)
X_all
```

[ ]:
```
['log_1',
 'sent_1',
```

```
 'log_2',
 'sent_2',
 'cluster_0_1',
 'cluster_0_2',
 'cluster_1_1',
 'cluster_1_2',
 'cluster_2_1',
 'cluster_2_2',
 'cluster_3_1',
 'cluster_3_2',
 'cluster_4_1',
 'cluster_4_2',
 'cluster_5_1',
 'cluster_5_2',
 'cluster_6_1',
 'cluster_6_2',
 'cluster_7_1',
 'cluster_7_2',
 'cluster_8_1',
 'cluster_8_2',
 'cluster_9_1',
 'cluster_9_2']
```

2.3.2 Question 3.2

Linearly regress SPY returns as a function of the lagged sentiment (2 lags) for each topic. This should be of the form $r_t = \beta_0 + \sum_{k=1}^{K}(\beta_{k,1}s_{k,t-1} + \beta_{k,2}s_{k,t-2})$ with K topics total. Evaluate the performance of this model with the mean squared error of the training data.

[ ]:
```
cols = X_all[4:]

X = df[cols]

lr2, mse2 = linreg(X)
```

```
  Intercept: 0.0034099293862093457

 cluster_0_1: -0.04368060661279426
 cluster_0_2: 0.005306805024557843
 cluster_1_1: -0.02532314214301729
 cluster_1_2: 0.0035390821675976313
 cluster_2_1: -0.004618287090389198
 cluster_2_2: 0.0013070076463209182
 cluster_3_1: -0.03397836845188566
 cluster_3_2: 0.019219671174599932
 cluster_4_1: 0.00012094105634543628
 cluster_4_2: -0.017480047476365494
```

```
  cluster_5_1: 0.006062232151662178
  cluster_5_2: -0.006305998425245751
  cluster_6_1: 0.012319830485283376
  cluster_6_2: -0.008357687897851367
  cluster_7_1: -0.05826020970623672
  cluster_7_2: 0.015010251067016667
  cluster_8_1: 0.014649675440873975
  cluster_8_2: -0.015446873701099852
  cluster_9_1: -0.024727419695585454
  cluster_9_2: -0.06301417067434663

MSE = 5.873463195465605e-05
```

2.3.3 Question 3.3

Linearly regress SPY returns as a function of the lagged returns and sentiment for each topic (2 lags each). This should be of the form $r_t = \beta_0 + \beta_{1,r} r_{t-1} + \beta_{2,r} r_{t-2} + \sum_{k=1}^{K} (\beta_{k,1,s} s_{k,t-1} + \beta_{k,2,s} s_{k,t-2})$ with K topics total. Evaluate the performance of this model with the mean squared error of the training data.

[ ]:
```python
X = [x for x in X_all if x not in ['sent_1', 'sent_2']]

lr3, mse3 = linreg(df[X])
```

```
    Intercept: 0.003342826903150288

        log_1: 0.03709925547610192
        log_2: 0.19104101073241453
  cluster_0_1: -0.05683016263096051
  cluster_0_2: 0.008730169724015437
  cluster_1_1: -0.033370820549430406
  cluster_1_2: 0.006106174567108706
  cluster_2_1: -0.004952829977398088
  cluster_2_2: 0.008274109715000021
  cluster_3_1: -0.03898957360633067
  cluster_3_2: 0.026959929334309348
  cluster_4_1: -0.004767614559797518
  cluster_4_2: -0.01635396102549187
  cluster_5_1: 0.012155045239428171
  cluster_5_2: -0.0037324283474021633
  cluster_6_1: 0.013797178927053393
  cluster_6_2: -0.01316457097060699
  cluster_7_1: -0.06806996279986878
  cluster_7_2: 0.010119542283322337
  cluster_8_1: 0.02218959374586673
  cluster_8_2: -0.009212673528608889
  cluster_9_1: -0.030146164476976458
  cluster_9_2: -0.061379559771011984
```

```
MSE = 5.65466064831002e-05
```

[ ]:
```python
# do one more linear regression with all the variables
lr4, mse4 = linreg(df[X_all])
```

```
    Intercept: 0.0030591796178134276

        log_1: 0.02857120115675503
       sent_1: 0.38674942714117283
        log_2: 0.2216032936819905
       sent_2: -0.1586331355061179
  cluster_0_1: -0.12373452012238063
  cluster_0_2: 0.036324554264927825
  cluster_1_1: -0.06716935547702339
  cluster_1_2: 0.021939955436803014
  cluster_2_1: -0.0435552468336734
  cluster_2_2: 0.026675652037434133
  cluster_3_1: -0.07843244347502183
  cluster_3_2: 0.044055687751873844
  cluster_4_1: -0.047667537947838866
  cluster_4_2: -0.0005847135591747391
  cluster_5_1: -0.014909938819301015
  cluster_5_2: 0.008807501525195167
  cluster_6_1: -0.03529473055470836
  cluster_6_2: 0.0017962565828821095
  cluster_7_1: -0.1225198939148208
  cluster_7_2: 0.03117785912239474
  cluster_8_1: -0.0048327606983597825
  cluster_8_2: -0.0030178099409436257
  cluster_9_1: -0.05679599234913632
  cluster_9_2: -0.04796822768550321

MSE = 5.512289512560037e-05
```

[ ]:
```python
# see with random forest
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor()
rf.fit(df[X_all], df['log'])

preds = rf.predict(df[X_all])

print("*" * 6, "|FEATURE IMPORTANCES|", "*" * 6)
for feat, imp in zip(df[X_all].columns, rf.feature_importances_):
    print(f"{feat:>13} : {imp}")
```

```
rf_mse = mean_squared_error(df['log'], preds)
print()
print(f"      MSE: {rf_mse}")
```

```
****** |FEATURE IMPORTANCES| ******
       log_1 : 0.018223593654228005
      sent_1 : 0.12483490714009944
       log_2 : 0.023537824999403732
      sent_2 : 0.025877496589269734
  cluster_0_1 : 0.02268366002912877
  cluster_0_2 : 0.03085370520019216
  cluster_1_1 : 0.025111184342087464
  cluster_1_2 : 0.021391833294298664
  cluster_2_1 : 0.020159626316692773
  cluster_2_2 : 0.026643605837924772
  cluster_3_1 : 0.025638679491116464
  cluster_3_2 : 0.02939705346056652
  cluster_4_1 : 0.017520522307446662
  cluster_4_2 : 0.06721384442730202
  cluster_5_1 : 0.04137818135949007
  cluster_5_2 : 0.05353019861670025
  cluster_6_1 : 0.023037469144374788
  cluster_6_2 : 0.011289276122054149
  cluster_7_1 : 0.0522710532171207
  cluster_7_2 : 0.015322696618652144
  cluster_8_1 : 0.07552840851318657
  cluster_8_2 : 0.046406809646948266
  cluster_9_1 : 0.07275337569462464
  cluster_9_2 : 0.12939499397709114

      MSE: 1.767404789961455e-05
```

2.3.4 Question 3.4

Compare the performance of these 3 linear regressions.

[ ]:
```
print(f"{mse1 = }")
print(f"{mse2 = }")
print(f"{mse3 = }")
print()
print(f"{mse4 = }")
print(f"{rf_mse = }")
```

```
mse1 = 0.00011083584269359552
mse2 = 5.873463195465605e-05
mse3 = 5.65466064831002e-05

mse4 = 5.512289512560037e-05
```

```
rf_mse = 1.767404789961455e-05
```

You can see that the best performing model was the one with both the lagged returns as well as the lagged cluster sentiments as well. The worst was the one that just had lagged returns. The MSE can get even better if you take an average sentiment for each day on top of the clusters. Finally, the random forest does significantly better than all of the linear regression which suggests that a more complicated model will do better than the multiple linear regressions.

2.4 Question 4 (10pt)

2.4.1 Question 4.1

Compare the performance of the various regressions utilized to those in prior homeworks. Do you find that applying topic modeling is beneficial in your analysis? Explain why or why not.

Topic modeling is definitely useful in my analysis. Comparing the results to homework 4 (pos, neg, neutral semtiments) the error is decreased slighly when looking at the linear regression. It is not a massive difference but it becomes larger when I add the average sentiment for each day in as well. This error could be decreased even more if I were to add a pos, neg, neutral sentiment in as well as they tended to do better than the aggregated score.

In conclusion, topic modeling is definitely beneficial in analysis but is one piece of the puzzle: when combining it with other parts as well it becomes even stronger and more accurate.