PREDICT THE RETURN OF STOCKS

IBM ADVANCED DATA SCIENCE SPECIALIZATION CAPSTONE PROJECT

PAUL LI 2019/10

USE CASE: PREDICT THE RETURN OF STOCKS

- Knowing the future return of stocks would be the best way of getting rich
- However this is a extremely hard task
- Our goal is to use public data to provide useful suggestions of investing in stocks
- In this project, we will use the stock TSE:L (Loblaw Companies Limited) to test the idea

DATA SET: STOCKS, ECONOMICS, AND GOOGLE TREND

STOCKS PRICE & VOLUME

OF COMPETITORS AND COMPOSITE INDICES

ECONOMICAL INDICES INCLUDE CPI, BCPI, INTEREST RATE, EXCHANGE RATE

GOOGLE TREND

DATA QUALITY ASSESSMENT: MISSING VALUES

```
data_extracted.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 5495 entries, 1999-01-01 to 2019-10-18
Data columns (total 21 columns):
loblaw_price           5019 non-null float64
loblaw_volume          5019 non-null float64
metro_price            5020 non-null float64
metro_volume           5020 non-null float64
empa_price             5020 non-null float64
empa_volume            5020 non-null float64
gwl_price              5019 non-null float64
gwl_volume             5019 non-null float64
atd_price              5020 non-null float64
atd_volume             5020 non-null float64
tsx_price              5020 non-null float64
tsx_volume             5020 non-null float64
sp500_price            4981 non-null float64
sp500_volume           4981 non-null float64
BCPI                   1030 non-null float64
CPI                    237 non-null float64
bank_interest          1030 non-null float64
CEER                   5421 non-null float64
trend_grocery_store    190 non-null float64
trend_loblaws          190 non-null float64
trend_stock            190 non-null float64
dtypes: float64(21)
memory usage: 944.5 KB
```

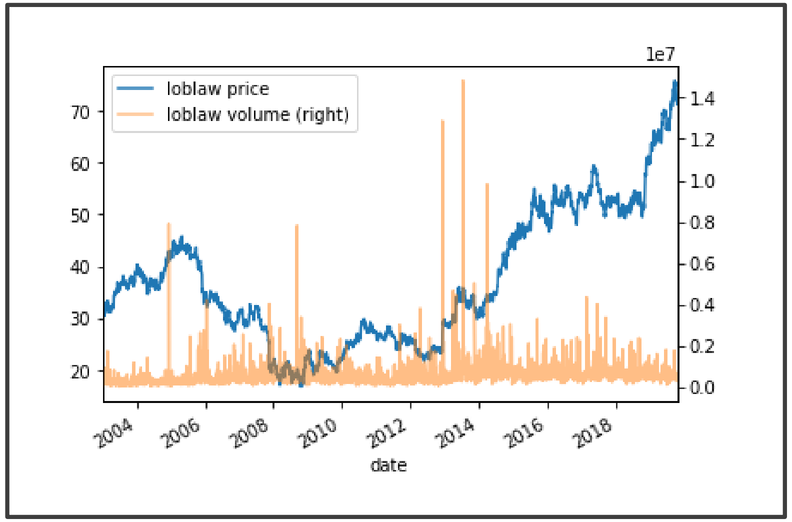
```
data_extracted.describe()
```

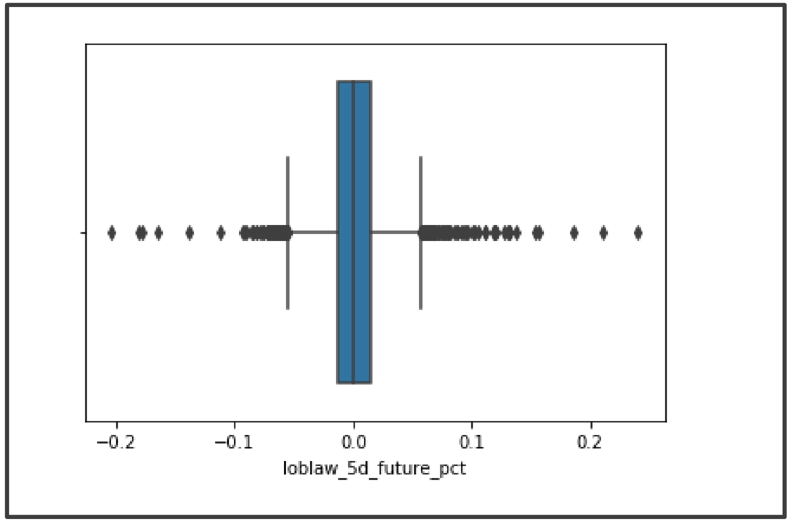
	loblaw_price	loblaw_volume	metro_price	metro_volume
count	5019.000000	5.019000e+03	5020.000000	5.020000e+03
mean	35.579635	5.293163e+05	16.246786	7.024548e+05
std	12.863246	5.922035e+05	15.055826	8.051032e+05
min	0.000000	0.000000e+00	1.101700	0.000000e+00
25%	26.289250	2.512000e+05	4.063425	3.465000e+05
50%	32.522400	4.141000e+05	8.986750	5.455500e+05
75%	44.048600	6.444000e+05	23.034000	8.541750e+05
max	75.770000	1.482520e+07	58.520000	3.157200e+07

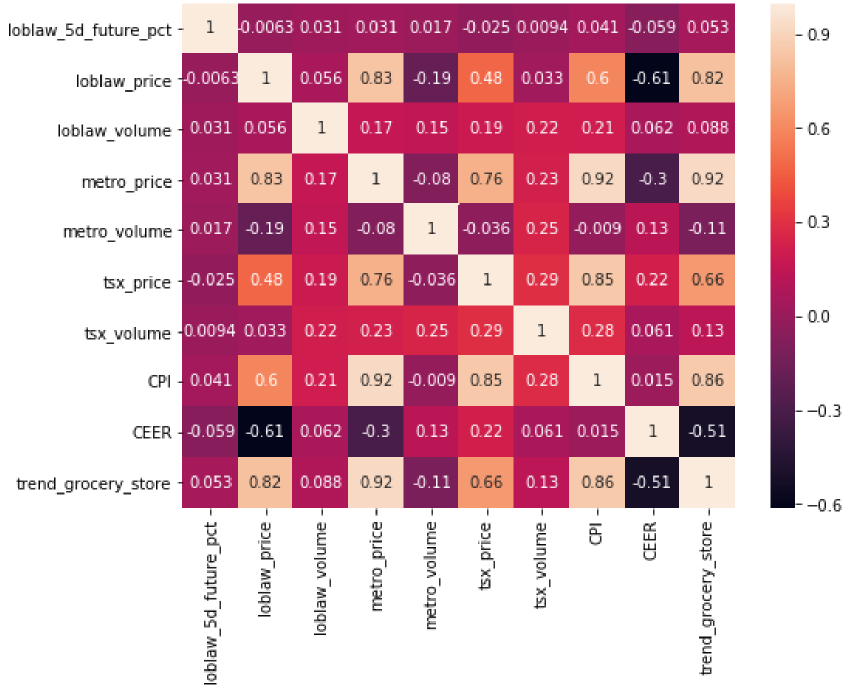
date	loblaw_price	loblaw_volume	metro_price	metro_volume
1999-01-01	NaN	NaN	NaN	NaN
1999-01-04	NaN	NaN	NaN	NaN
1999-01-05	NaN	NaN	NaN	NaN
1999-01-06	NaN	NaN	NaN	NaN
1999-01-07	NaN	NaN	NaN	NaN

- Removal
- Interpolation

DATA EXPLORATION AND VISUALIZATION







- Fluctuation
- Distribution
- Small Correlations

FEATURE ENGINEERING

qcut = 5

```
df['loblaw_5d_future'] = df['loblaw_price'].shift(-5)
df['loblaw_5d_future_pct'] = df['loblaw_5d_future'].pct_change(5)
df['loblaws_5d_return_level'] = pd.qcut(df['loblaw_5d_future_pct'], 5, labels=False)
```

outliers

```
Q1 = df['loblaw_5d_future_pct'].quantile(0.25)
Q3 = df['loblaw_5d_future_pct'].quantile(0.75)
IQR = Q3 - Q1

df['loblaws_5d_return_level'] = df['loblaw_5d_future_pct'].apply(lambda x: 0 if x < (Q1 - 1.5 * IQR)
                                                                   else 1 if (Q1 - 1.5 * IQR) < x < Q1
                                                                   else 2  if Q1 < x < Q3
                                                                   else 3  if Q3 < x < (Q3 + 1.5 * IQR)
                                                                   else 4)
```

```
feature_names = []
stock_list = ['loblaw', 'metro', 'gwl', 'empa', 'atd', 'tsx', 'sp500']
for stock in stock_list:
    for n in [14, 30, 50, 200]:
        df[stock + '_ma' + str(n)] = talib.SMA(df['loblaw_price'].values, timeperiod=n) / df['loblaw_price']
        df[stock + '_rsi' + str(n)] = talib.RSI(df['loblaw_price'].values, timeperiod=n)
        feature_names = feature_names + [stock + '_ma' + str(n), stock + '_rsi' + str(n)]

    df[stock + '_5d_pct'] = df[stock + '_price'].pct_change(5)
    feature_names = feature_names + [stock + '_5d_pct']
    feature_names = feature_names + [stock + '_price']

    df[stock + '_volume_1d_pct_SMA'] = talib.SMA(df[stock + '_volume'].pct_change().values, timeperiod=5)
    feature_names = feature_names + [stock + '_volume_1d_pct_SMA']
```

```
economic_indecis = ['BCPI', 'CPI', 'bank_interest', 'CEER']
google_trends = ['trend_grocery_store', 'trend_loblaws', 'trend_stock']
feature_names = feature_names + economic_indecis + google_trends
```

Simple Moving Average (SMA)

$$\text{SMA} = \frac{A_1 + A_2 + \ldots + A_n}{n}$$

where:

A_n = the price of an asset at period n

n = the number of total periods

Relative Strength Index – RSI

$$RSI_{\text{step one}} = 100 - \left[\frac{100}{1 + \frac{\text{Average gain}}{\text{Average loss}}} \right]$$

RANDOM FOREST CLASSIFIER

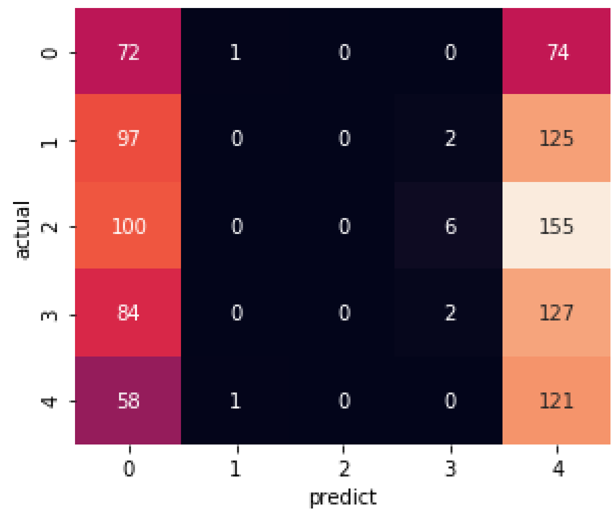
qcut = 5

```
df['loblaw_5d_future'] = df['loblaw_price'].shift(-5)
df['loblaw_5d_future_pct'] = df['loblaw_5d_future'].pct_change(5)
df['loblaws_5d_return_level'] = pd.qcut(df['loblaw_5d_future_pct'], 5, labels=False)
```

```
from sklearn.model_selection import ParameterGrid
from sklearn.ensemble import RandomForestClassifier
```

```
grid = {'n_estimators':[200], 'max_depth': [3,5,7], 'max_features': [4,8,16,32,64], 'random_state': [42]}
test_scores = []
rfc = RandomForestClassifier()

for g in ParameterGrid(grid):
    rfc.set_params(**g)
    rfc.fit(train_features, train_targets)
    test_scores.append(rfc.score(test_features, test_targets))

best_idx = np.argmax(test_scores)
print(test_scores[best_idx], ParameterGrid(grid)[best_idx])
```

```
0.1902439024390244 {'random_state': 42, 'n_estimators': 200, 'max_features': 64, 'max_depth': 3}
```



RANDOM FOREST CLASSIFIER – CONT'D

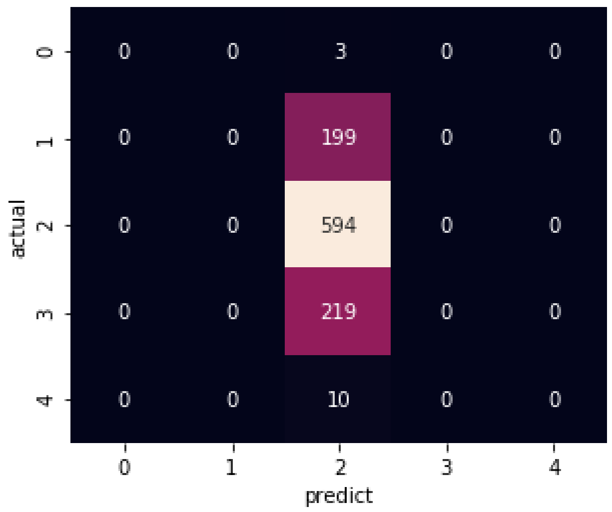
outliers

```
Q1 = df['loblaw_5d_future_pct'].quantile(0.25)
Q3 = df['loblaw_5d_future_pct'].quantile(0.75)
IQR = Q3 - Q1

df['loblaws_5d_return_level'] = df['loblaw_5d_future_pct'].apply(lambda x: 0 if x < (Q1 - 1.5 * IQR)
                                                              else 1 if (Q1 - 1.5 * IQR) < x < Q1
                                                              else 2  if Q1 < x < Q3
                                                              else 3  if Q3 < x < (Q3 + 1.5 * IQR)
                                                              else 4)
```

```
from sklearn.model_selection import ParameterGrid
from sklearn.ensemble import RandomForestClassifier
```

```
grid = {'n_estimators':[200], 'max_depth': [3,5,7], 'max_features': [4,8,16,32,64], 'random_state': [42]}
test_scores = []
rfc = RandomForestClassifier()

for g in ParameterGrid(grid):
    rfc.set_params(**g)
    rfc.fit(train_features, train_targets)
    test_scores.append(rfc.score(test_features, test_targets))

best_idx = np.argmax(test_scores)
print(test_scores[best_idx], ParameterGrid(grid)[best_idx])
```

```
0.5795121951219512 {'random_state': 42, 'n_estimators': 200, 'max_features': 4, 'max_depth': 3}
```



RANDOM FOREST CLASSIFIER – CONT'D

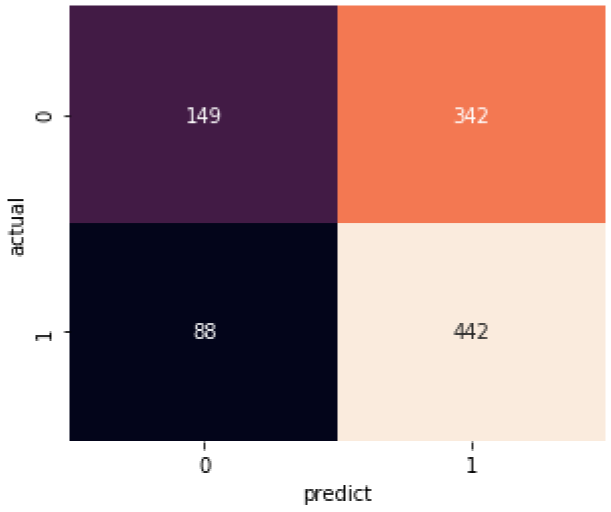
qcut = 2
20d return in the future

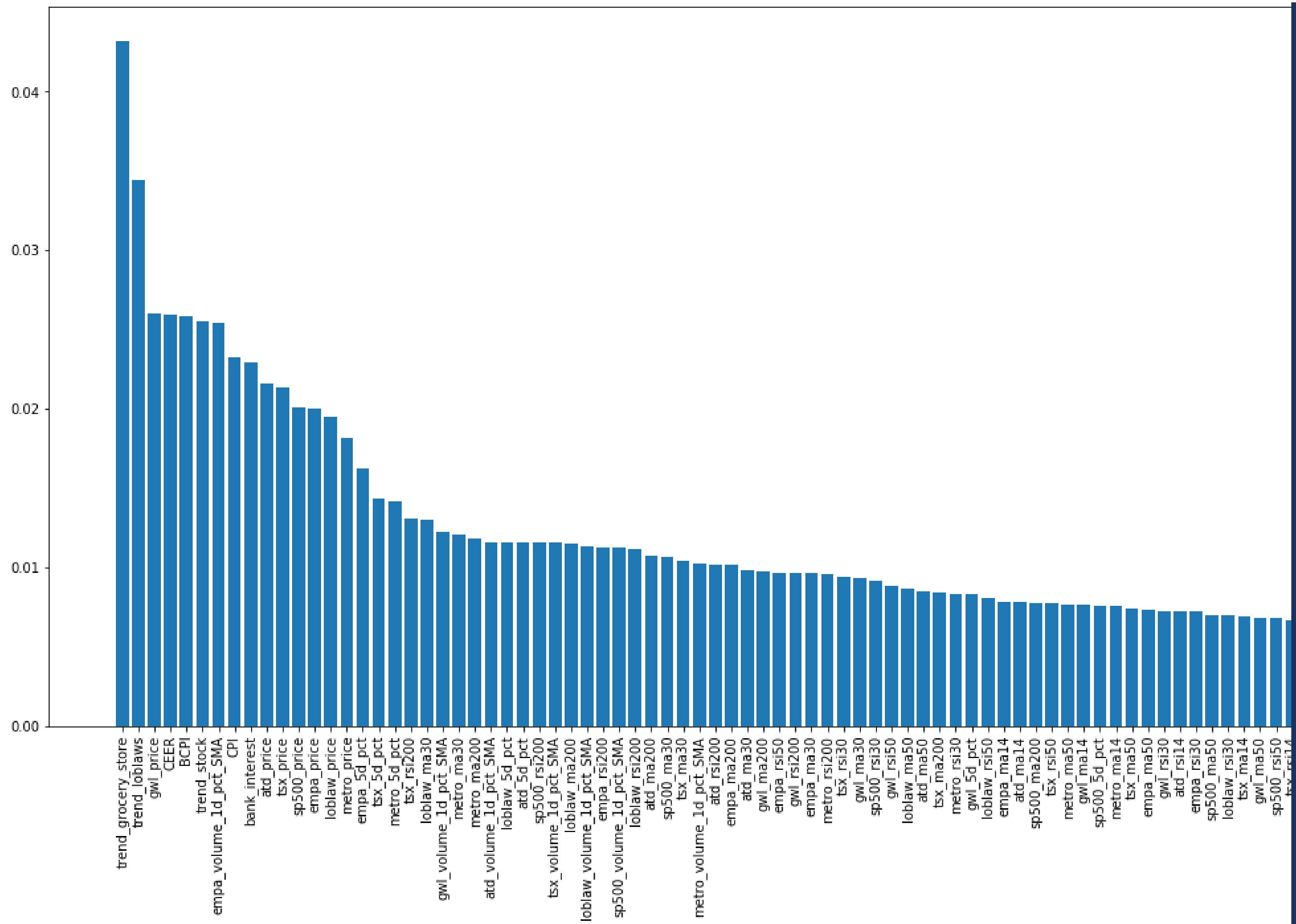
```
df['loblaw_20d_future'] = df['loblaw_price'].shift(-20)
df['loblaw_20d_future_pct'] = df['loblaw_20d_future'].pct_change(20)
df['loblaws_20d_return_level'] = pd.qcut(df['loblaw_20d_future_pct'], 2, labels=False)
```

```
from sklearn.model_selection import ParameterGrid
from sklearn.ensemble import RandomForestClassifier
```

```
grid = {'n_estimators':[200], 'max_depth': [3,5,7], 'max_features': [4,8,16,32,64], 'random_state': [42]}
test_scores = []
rfc = RandomForestClassifier()

for g in ParameterGrid(grid):
    rfc.set_params(**g)
    rfc.fit(train_features, train_targets)
    test_scores.append(rfc.score(test_features, test_targets))

best_idx = np.argmax(test_scores)
print(test_scores[best_idx], ParameterGrid(grid)[best_idx])
```

```
0.5788442703232125 {'random_state': 42, 'n_estimators': 200, 'max_features': 8, 'max_depth': 7}
```





FEATURE SELECTION

DEEP LEARNING - LSTM

```
from sklearn.preprocessing import scale
scaled_features = scale(features)
```

```
scaled_features.shape
```

```
(4084, 84)
```

```
batch_size = 64
timesteps = 20
```

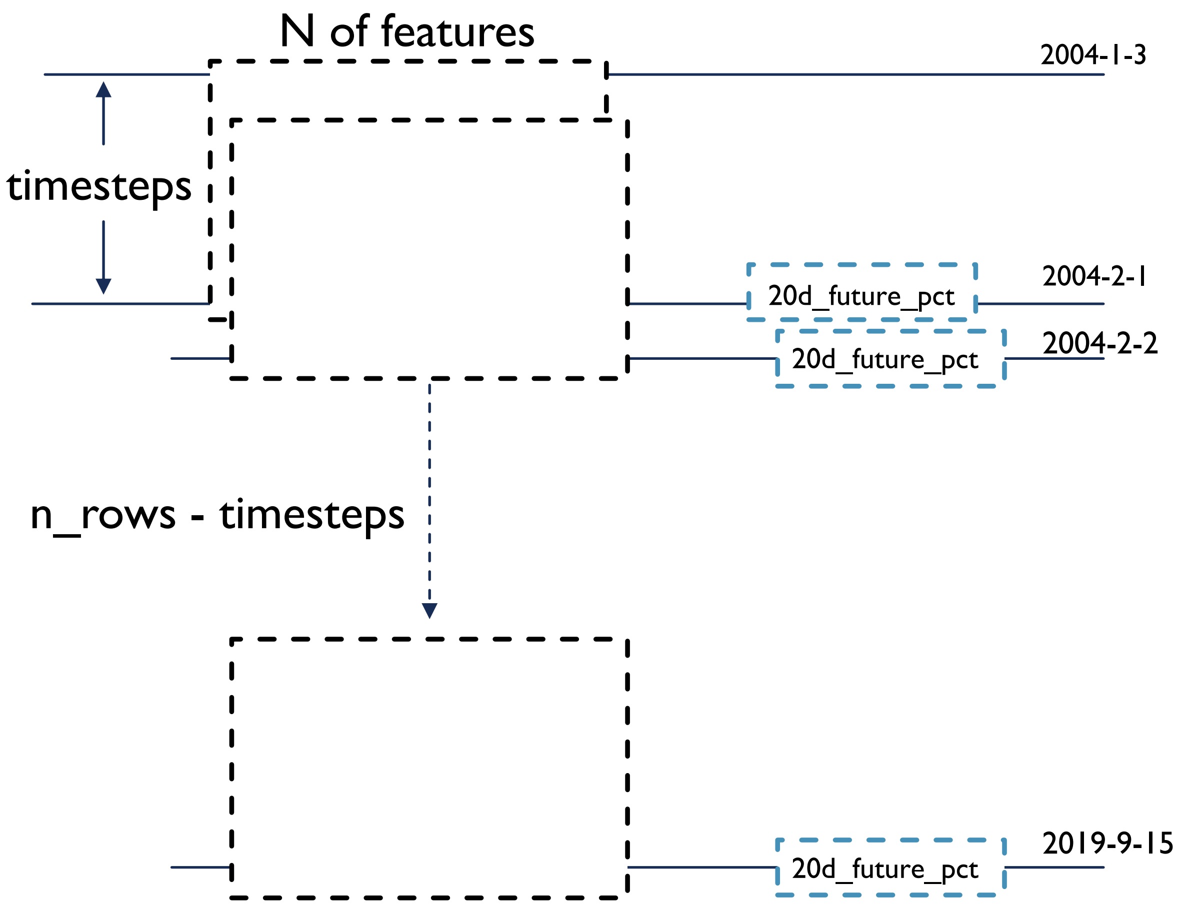
```
scaled_lstm_features = []
for column in range(scaled_features.shape[1]):
    time_series = []
    for i in range(len(scaled_features) - timesteps):
        time_series.append(scaled_features[i:i + timesteps, column])
    scaled_lstm_features.append(time_series)
scaled_lstm_features = np.dstack(scaled_lstm_features)
print(type(scaled_lstm_features))
print(scaled_lstm_features.shape)
```

```
<class 'numpy.ndarray'>
(4064, 20, 84)
```

```
lstm_targets = targets[timesteps:]
```

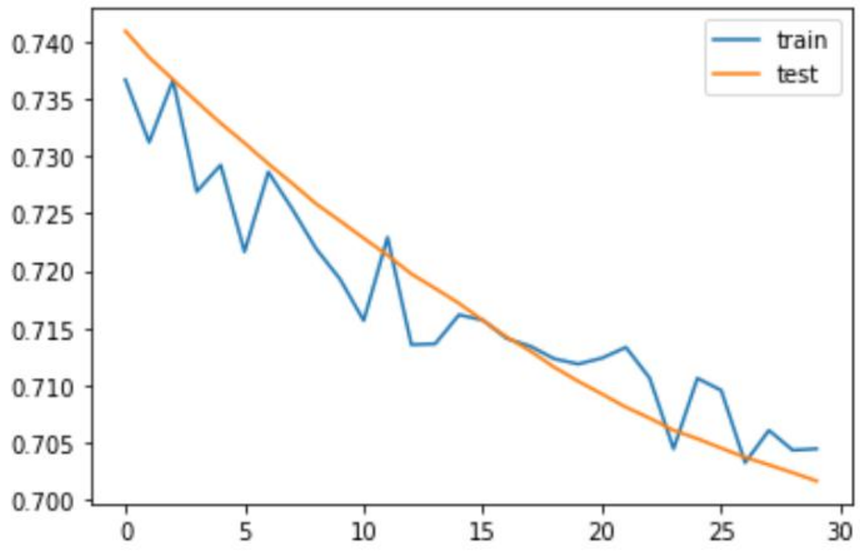
```
lstm_targets.shape
```

```
(4064,)
```



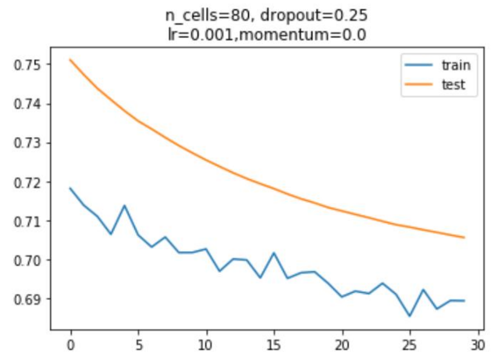
DEEP LEARNING – LSTM – CONT'D

```
n_cells = 20
dropout = 0.25
lr = 0.001
momentum = 0.0
epochs = 30

model = Sequential()
model.add(LSTM(n_cells, input_shape=(scaled_train_features.shape[1], scaled_train_features.shape[2]),
               dropout=dropout, recurrent_dropout=dropout))
model.add(Dense(1, activation='sigmoid'))

sgd = optimizers.SGD(lr = lr, momentum=momentum)
model.compile(loss='binary_crossentropy', optimizer=sgd)

history = model.fit(scaled_train_features, train_targets, epochs=epochs, batch_size=batch_size,
                    validation_data=(scaled_test_features, test_targets), verbose=0, shuffle=False)

plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()

test_pred = model.predict(scaled_test_features)

print(confusion_matrix(test_targets, test_pred > 0.5))
print(accuracy_score(test_targets, test_pred > 0.5))
```

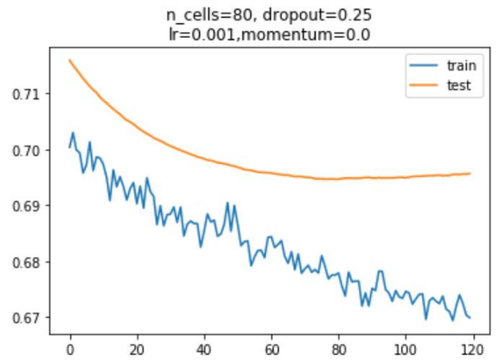



```
[[196 290]
 [192 338]]
0.5255905511811023
```

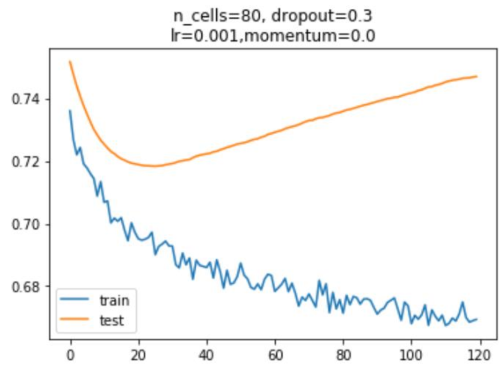
DEEP LEARNING – LSTM – CONT'D



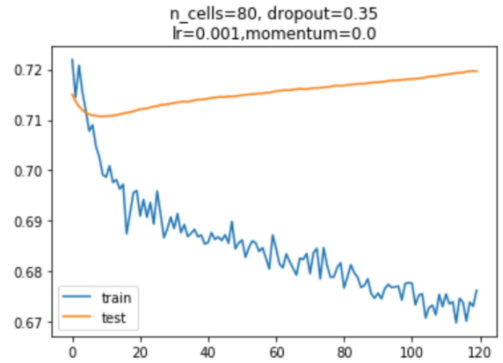

```
confusionmatrix:
 [[404  82]
 [433  97]]
accuracy score:  0.49311023622047245
```



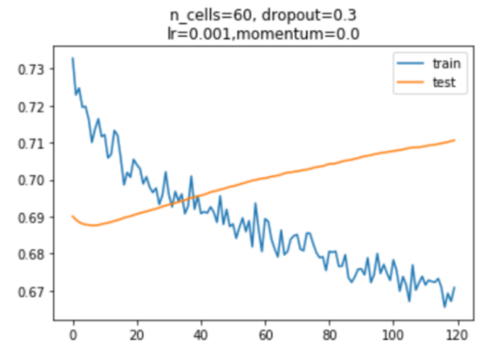

```
confusionmatrix:
 [[ 93 393]
 [ 89 441]]
accuracy score:  0.5255905511811023
```



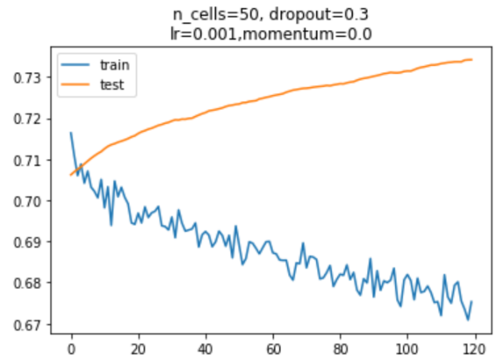

```
confusionmatrix:
 [[ 20 466]
 [ 10 520]]
accuracy score:  0.531496062992126
```



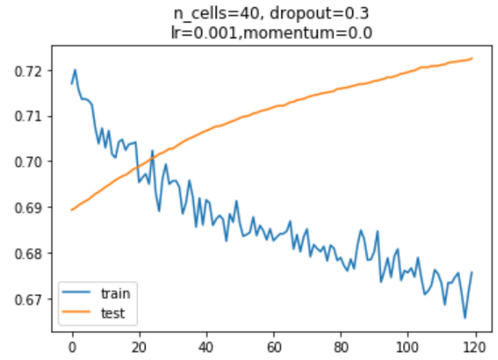

```
confusionmatrix:
 [[ 56 430]
 [ 69 461]]
accuracy score:  0.5088582677165354
```



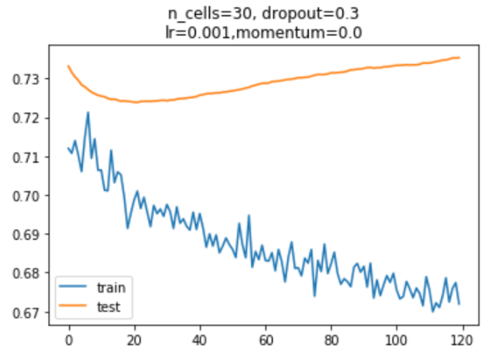

```
confusionmatrix:
 [[ 76 410]
 [ 64 466]]
accuracy score:  0.5334645669291339
```




```
confusionmatrix:
 [[  7 479]
 [  6 524]]
accuracy score:  0.5226377952755905
```

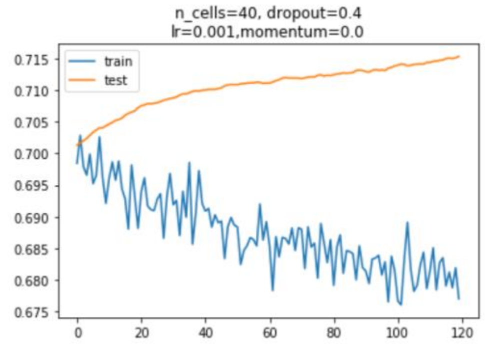



```
confusionmatrix:
 [[ 19 467]
 [  0 530]]
accuracy score:  0.5403543307086615
```

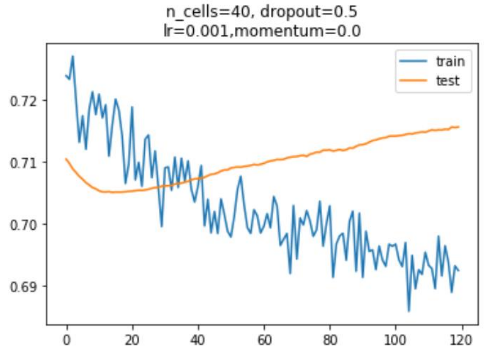



```
confusionmatrix:
 [[ 17 469]
 [ 16 514]]
accuracy score:  0.5226377952755905
```

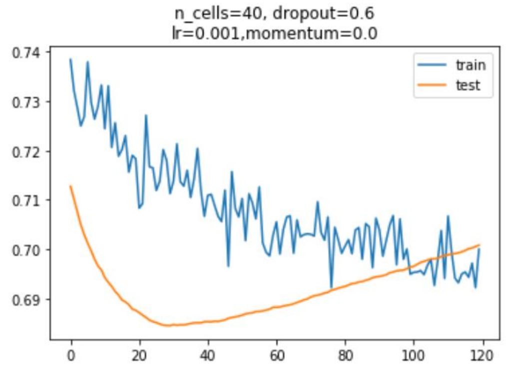
DEEP LEARNING –LSTM – CONT'D



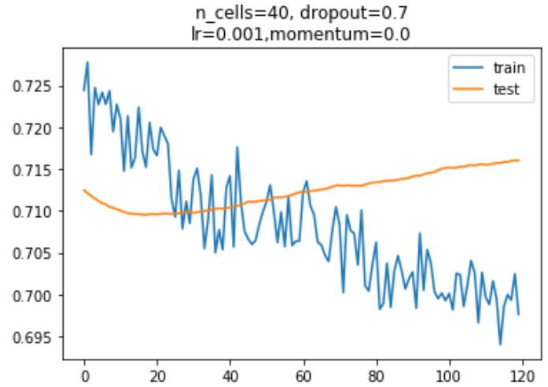

```
confusionmatrix:
 [[ 21 465]
 [ 11 519]]
accuracy score:  0.531496062992126
```



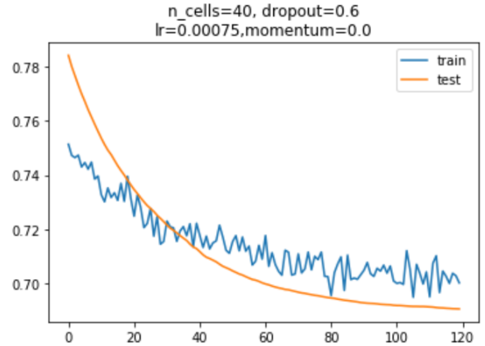

```
confusionmatrix:
 [[ 41 445]
 [ 31 499]]
accuracy score:  0.531496062992126
```



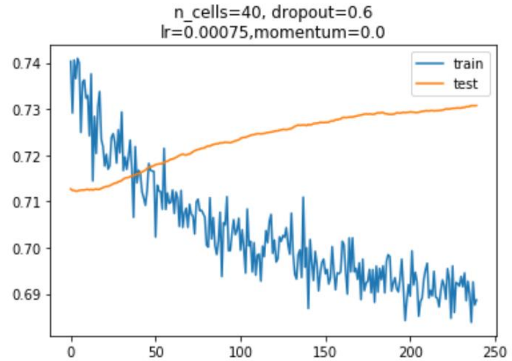

```
confusion matrix:
 [[ 58 428]
 [ 30 500]]
accuracy score:  0.5492125984251969
```



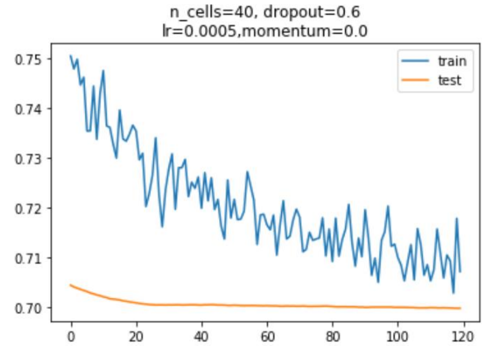

```
confusion matrix:
 [[121 365]
 [117 413]]
accuracy score:  0.5255905511811023
```



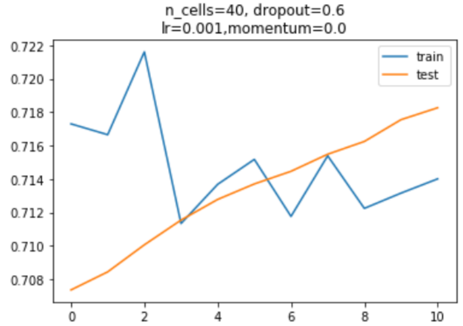

```
confusion matrix:
 [[117 369]
 [124 406]]
accuracy score:  0.514763779527559
```




```
confusion matrix:
 [[ 30 456]
 [ 36 494]]
accuracy score:  0.515748031496063
```




```
confusion matrix:
 [[141 345]
 [182 348]]
accuracy score:  0.4812992125984252
```




```
confusion matrix:
 [[ 50 436]
 [ 32 498]]
accuracy score:  0.5393700787401575
```

EarlyStopping(monitor='val_loss', mode='min',patience=10)

NEXT STEP

- Improve data quality
- Schedule the task
- More research about LSTM

THANK YOU

PAULLIMALE@HOTMAIL.COM

https://github.com/paulkillus/stock_return