Predicting the direction of stock market prices using random forest

Abhishek Singh 201751002

Vikash Kumar 201751061

Ranjeet Singh 201751041

Alok Kumar 201752008

Sahadev Indalia 201751042

In [18]:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (7,4.5) # Make the default figures a bit bigger

import numpy as np
import random

#Let's make this notebook reproducible
np.random.seed(42)
random.seed(42)

import pandas_techinal_indicators as ta #https://github.com/Crypto-toolbox/pandas-techn
ical-indicators/blob/master/technical_indicators.py

import pandas as pd
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import f1_score, precision_score, confusion_matrix, recall_score,
accuracy_score
from sklearn.model_selection import train_test_split
```

In [19]:

```
aapl = pd.read_csv('AAPL.csv')
del(aapl['Date'])
del(aapl['Adj Close'])
aapl.head()
```

Out[19]:

	Open	High	Low	Close	Volume
0	33.641430	33.801430	33.494286	33.571430	107664900
1	33.915714	34.104286	33.250000	33.709999	150786300
2	33.568573	34.072857	33.538570	34.070000	171126900
3	34.028572	34.320000	33.857143	34.220001	111754300
4	34.221428	34.560001	34.094284	34.371429	157125500

In [20]:

```
def get_exp_preprocessing(df, alpha=0.9):
    edata = df.ewm(alpha=alpha).mean()
    return edata
```

In [21]:

```
saapl = get_exp_preprocessing(aapl)
saapl.head() #saapl stands for smoothed aapl
```

Out[21]:

	Open	High	Low	Close	Volume
0	33.641430	33.801430	33.494286	33.571430	1.076649e+08
1	33.890779	34.076754	33.272208	33.697402	1.468662e+08
2	33.600503	34.073243	33.512174	34.033076	1.687227e+08
3	33.985804	34.295347	33.822677	34.201325	1.174460e+08
4	34.197868	34.533538	34.067126	34.354420	1.531579e+08

In [22]:

```
def feature_extraction(data):
    for x in [5, 14, 26, 44, 66]:
        data = ta.relative_strength_index(data, n=x)
        data = ta.stochastic_oscillator_d(data, n=x)
        data = ta.accumulation_distribution(data, n=x)
        data = ta.average_true_range(data, n=x)
        data = ta.momentum(data, n=x)
        data = ta.money_flow_index(data, n=x)
        data = ta.rate_of_change(data, n=x)
        data = ta.on_balance_volume(data, n=x)
        data = ta.commodity_channel_index(data, n=x)
        data = ta.ease_of_movement(data, n=x)
        data = ta.trix(data, n=x)
        data = ta.vortex_indicator(data, n=x)

    data['ema50'] = data['Close'] / data['Close'].ewm(50).mean()
    data['ema21'] = data['Close'] / data['Close'].ewm(21).mean()
    data['ema14'] = data['Close'] / data['Close'].ewm(14).mean()
    data['ema5'] = data['Close'] / data['Close'].ewm(5).mean()

    #Williams %R is missing
    data = ta.macd(data, n_fast=12, n_slow=26)

    del(data['Open'])
    del(data['High'])
    del(data['Low'])
    del(data['Volume'])

    return data

def compute_prediction_int(df, n):
    pred = (df.shift(-n)['Close'] >= df['Close'])
    pred = pred.iloc[:-n]
    return pred.astype(int)

def prepare_data(df, horizon):
    data = feature_extraction(df).dropna().iloc[:-horizon]
    data['pred'] = compute_prediction_int(data, n=horizon)
    del(data['Close'])
    return data.dropna()
```

In [23]:

```
data = prepare_data(saapl, 10)

y = data['pred']

#remove the output from the input
features = [x for x in data.columns if x not in ['gain', 'pred']]
X = data[features]
```

In [24]:

```
train_size = 2*len(X) // 3

X_train = X[:train_size]
X_test = X[train_size:]
y_train = y[:train_size]
y_test = y[train_size:]
```

In [25]:

```
print('len X_train', len(X_train))
print('len y_train', len(y_train))
print('len X_test', len(X_test))
print('len y_test', len(y_test))
```

```
len X_train 644
len y_train 644
len X_test 323
len y_test 323
```

In [26]:

```
rf = RandomForestClassifier(n_jobs=-1, n_estimators=65, random_state=42)
rf.fit(X_train, y_train.values.ravel());
```

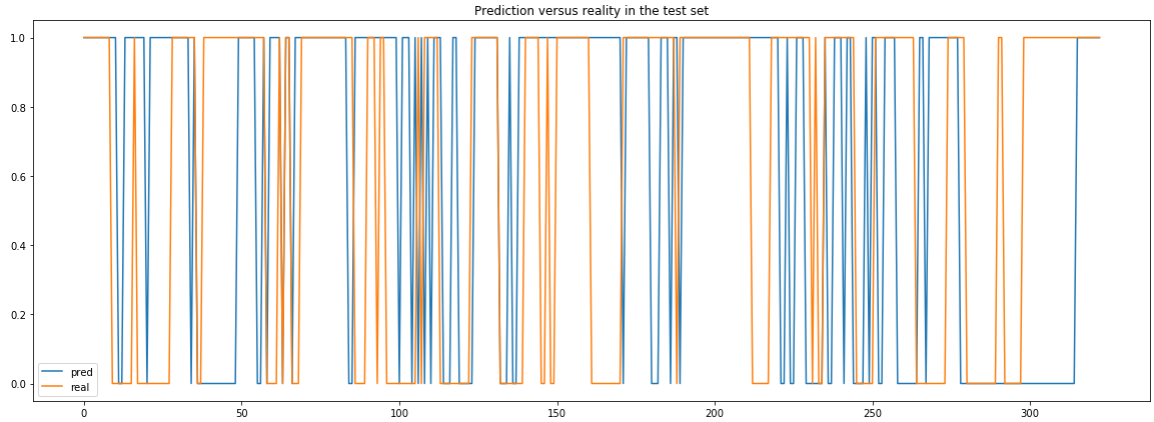
In [27]:

```
pred = rf.predict(X_test)
precision = precision_score(y_pred=pred, y_true=y_test)
recall = recall_score(y_pred=pred, y_true=y_test)
f1 = f1_score(y_pred=pred, y_true=y_test)
accuracy = accuracy_score(y_pred=pred, y_true=y_test)
confusion = confusion_matrix(y_pred=pred, y_true=y_test)
print('precision: {0:1.2f}, recall: {1:1.2f}, f1: {2:1.2f}, accuracy: {3:1.2f}'.format(
precision, recall, f1, accuracy))
print('Confusion Matrix')
print(confusion)
```

```
precision: 0.66, recall: 0.68, f1: 0.67, accuracy: 0.58
Confusion Matrix
[[ 47  71]
 [ 66 139]]
```

In [28]:

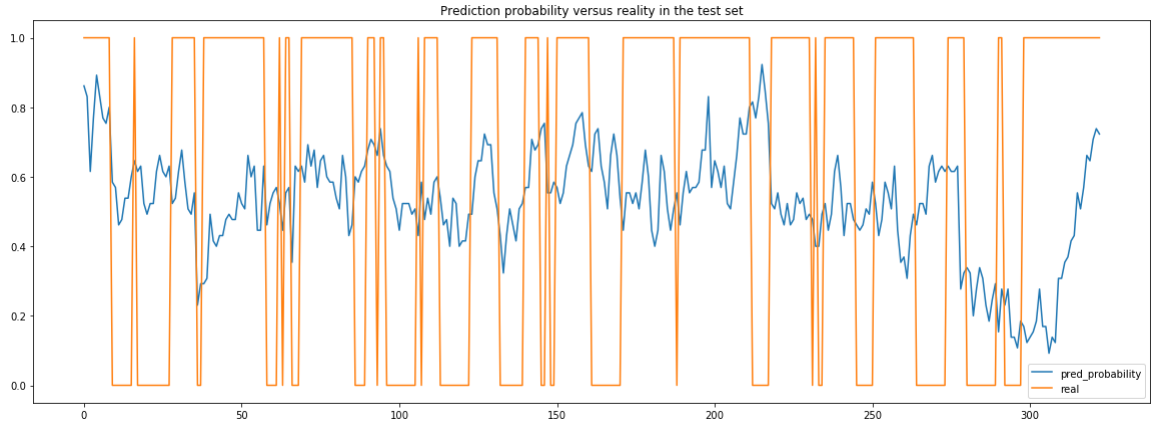
```
plt.figure(figsize=(20,7))
plt.plot(np.arange(len(pred)), pred, label='pred')
plt.plot(np.arange(len(y_test)), y_test, label='real' );
plt.title('Prediction versus reality in the test set')
plt.legend();
```



In [29]:

```
plt.figure(figsize=(20,7))
proba = rf.predict_proba(X_test)[:,1]
plt.figure(figsize=(20,7))
plt.plot(np.arange(len(proba)), proba, label='pred_probability')
plt.plot(np.arange(len(y_test)), y_test, label='real' );
plt.title('Prediction probability versus reality in the test set');
plt.legend();
plt.show();
```

```
<Figure size 1440x504 with 0 Axes>
```



In [30]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 2*len(X) // 3)

print('len X_train', len(X_train))
print('len y_train', len(y_train))
print('len X_test', len(X_test))
print('len y_test', len(y_test))
```

```
len X_train 644
len y_train 644
len X_test 323
len y_test 323
```

In [31]:

```
rf = RandomForestClassifier(n_jobs=-1, n_estimators=65, random_state=42)
rf.fit(X_train, y_train.values.ravel());
```

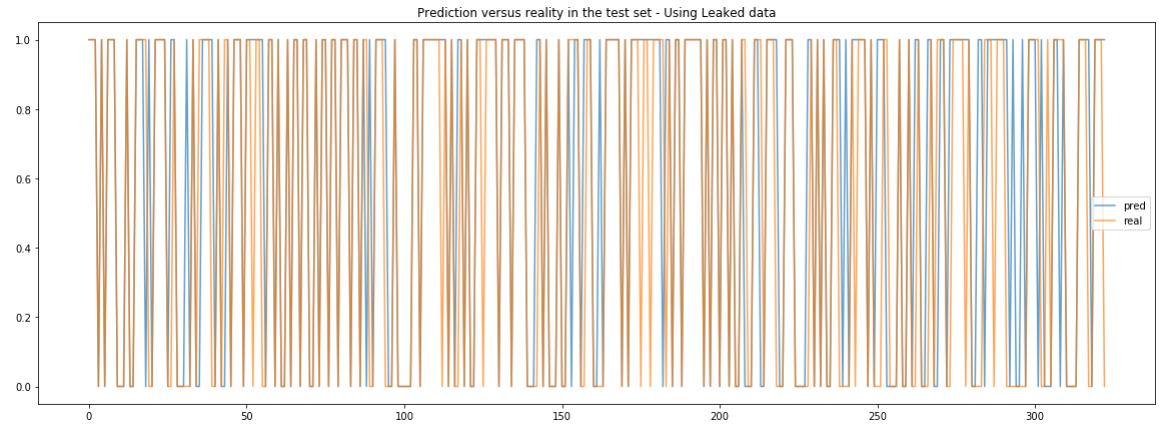
In [32]:

```
pred = rf.predict(X_test)
precision = precision_score(y_pred=pred, y_true=y_test)
recall = recall_score(y_pred=pred, y_true=y_test)
f1 = f1_score(y_pred=pred, y_true=y_test)
accuracy = accuracy_score(y_pred=pred, y_true=y_test)
confusion = confusion_matrix(y_pred=pred, y_true=y_test)
print('precision: {0:1.2f}, recall: {1:1.2f}, f1: {2:1.2f}, accuracy: {3:1.2f}'.format(
precision, recall, f1, accuracy))
print('Confusion Matrix')
print(confusion)
```

```
precision: 0.81, recall: 0.92, f1: 0.86, accuracy: 0.84
Confusion Matrix
[[110  37]
 [ 14 162]]
```

In [33]:

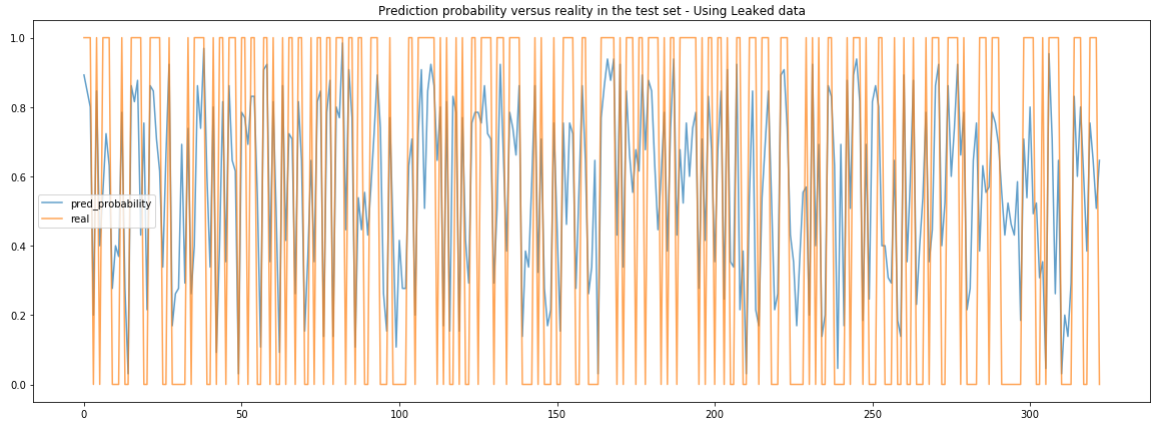
```
plt.figure(figsize=(20,7))
plt.plot(np.arange(len(pred)), pred, alpha=0.7, label='pred')
plt.plot(np.arange(len(y_test)), y_test, alpha=0.7, label='real' );
plt.title('Prediction versus reality in the test set - Using Leaked data')
plt.legend();
```



In [34]:

```
plt.figure(figsize=(20,7))
proba = rf.predict_proba(X_test)[:,1]
plt.figure(figsize=(20,7))
plt.plot(np.arange(len(proba)), proba, alpha = 0.7, label='pred_probability')
plt.plot(np.arange(len(y_test)), y_test, alpha = 0.7, label='real' );
plt.title('Prediction probability versus reality in the test set - Using Leaked data');
plt.legend();
plt.show();
```

```
<Figure size 1440x504 with 0 Axes>
```



In [ ]:

In [ ]: