Project_Gabriel_Astieres

April 30, 2021

1 Introduction

I'm very interested by cryptocurrencies and started to dive in that universe one year ago. I wanted to start building a trading bot and used this class project as a opportunity to get started.

The idea will be to scrap everyday the marked price of the market, plug that into my Neural Network and take a position according to the prediction via an API to an coin exchange plateform.

The project is subdivided in 4 main parts: Feature Engineering, Dense Layer Model, Decision Tree Classifier and finally a LTSM CNN. Each time you will find a detailled explanation of the section in a markdown at its beginning.

In order to have to obtain promessing results, I made a lot of research and trials. However, to keep this as short as possible, only the final methods I kept will be shown and detailled here.

[1]:
```
import tensorflow as tf
import pandas as pd
import numpy as np
from tensorflow.keras.models import *
from tensorflow.keras.layers import *

import sklearn.decomposition as sk
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report
from sklearn import tree
from sklearn import preprocessing

import seaborn as sns
from matplotlib import pyplot as plt

import plotly.graph_objects as go

#Technical analysis of charts library
import ta
# help(ta)
```

2 Part I: Feature Engineering

The key element to obtain viable results lies in the feature and the data themselves. I have a Dataset for each of the most importants coins, each of them cointains the Open, Close, High and Low values in USD for everyday. In the future, those values will be farely easy to access on a daily basis using an API to any reliable website.

I created a function "Pre_Processing" that would proceed to any pretreatment required and output a useable dataset. I made the function in such a way that I could use it on Ethereum, Bitcoin or any other coin and still obtain an appropriate result with the good columns name and everything in place. You can call it like this: df = Pre_Processing("NameOfTheCurrency").

The few main steps this function follows are: the Normalization, the removing of useless columns, the creation of the label and the adding of a couple features such as the RSI (Relative Strengh Index) and the moving average over 14 and 60 periods of time. The label is the variation of the coin, if the next closing is higher than the previous we have 1, otherwise 0.

You can see in this part, the 3 Component PCA as well as the correlation heat map and notice that my features aren't really correlated with the label I implemented. I was struggling to obtain any promessing result like this.

Therefore, after some research I found a great library that computes most of the technical indicators of a price chart. You can compare this second dataset with the new PCA and the new correlation heatmap. This dataframe contains 90 features, all computed from the 4 inital ones, and that is what I used for the rest of the project.

[2]:
```
def Normalize(df):
#     Min-Max normalization
    normalized = (df-df.min())/(df.max()-df.min())

#     mean/std normalization
#     (df-df.mean())/df.std()
    return(normalized)


def Get_RSI(price, n=14):

    delta = price.diff()
    dUp, dDown = delta.copy(), delta.copy()

    dUp[dUp < 0] = 0
    dDown[dDown > 0] = 0

    RolUp =  dUp.rolling(n).mean()
    RolDown = dDown.rolling(n).mean().abs()

    RS = RolUp / RolDown
    RSI = 100 - (100 / (1 + RS))
    return(RSI)
```

```
def Pre_Processing(coin, previous = False, Use_TA_lib = True):
    # Read the file of the coin
    df = pd.read_csv('Data/coin_'+coin+'.csv')

    ticker = df["Symbol"][1]

#     # Take only last year
#     df = df.tail(365)

    # Drop useless columns
    df = df.drop('Name', axis='columns')
    df = df.drop('Symbol', axis='columns')
    df = df.drop('SNo', axis='columns')
    df = df.drop('Date', axis='columns')


    if Use_TA_lib:
        df = ta.add_all_ta_features(df, open="Open", high="High", low="Low",␣
↪close="Close", volume="Volume", fillna=True)

    else:
        # Add moving average 14 & 60 periods
        df['MA_14'] = df['Close'].rolling(window=14,center=False).mean()
        df['MA_60'] = df['Close'].rolling(window=60,center=False).mean()

        # Add MA Previous
        if previous:
            for i in range(1,8):
                df['MA_14_-' + str(i)] = df['MA_14'].shift(periods=-i,␣
↪freq=None, axis=0)
                df['MA_60_-' + str(i)] = df['MA_60'].shift(periods=-i,␣
↪freq=None, axis=0)

        # Add RSI
        df['RSI'] = Get_RSI(df['Close'])

        # Add RSI Previous
        if previous:
            for i in range(1,8):
                df['RSI_-' + str(i)] = Get_RSI(df['Close']).shift(periods=-i,␣
↪freq=None, axis=0)

    # Add +1 or -1 for variation
    diff = np.sign(df['Close'].diff(periods=1)).shift(periods=-1, freq=None,␣
↪axis=0)
    df["Variation"] = diff
```

```python
    # Normalize
    df = Normalize(df)

    names = []
    for column in df:names+=[column + '_' + ticker]
    df.columns = names

    # Drop lines with NaN
    df = df.dropna(axis='rows')

    # Reset the index to have a proper count
    df = df.reset_index()

    return(df)
```

[3]:

```python
# df = pd.read_csv('Data/coin_Ethereum.csv')
df = pd.read_csv('Data/coin_Bitcoin.csv')

fig = go.Figure(data=[go.Candlestick(x=df['Date'],
                open=df['Open'],
                high=df['High'],
                low=df['Low'],
                close=df['Close'])])
fig.show()




# This is an iteractive graph for illustration purpose, feel free to zoom on␣
↪any relevant period and manipulate it using the
# slider at the bottom or the top right tools.
```

[4]:

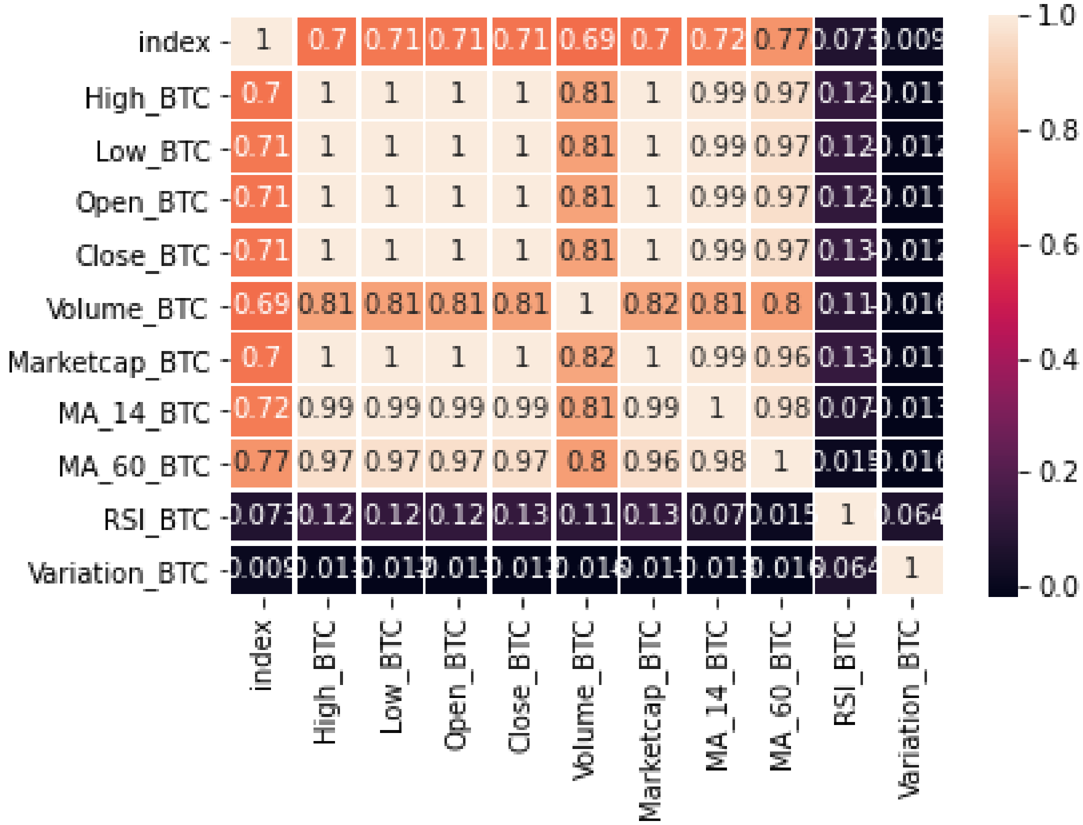
```python
# The DF using my method
df_My_Features = Pre_Processing("Bitcoin", False, False)
df_My_Features.head()
```

[4]:

```
   index  High_BTC   Low_BTC  Open_BTC  Close_BTC  Volume_BTC  Marketcap_BTC  \
0     59  0.000505  0.000640  0.000618   0.000574         0.0       0.000347
1     60  0.000466  0.000482  0.000578   0.000456         0.0       0.000276
2     61  0.000437  0.000494  0.000455   0.000462         0.0       0.000280
3     62  0.000404  0.000516  0.000461   0.000490         0.0       0.000297
4     63  0.000397  0.000374  0.000505   0.000341         0.0       0.000207

   MA_14_BTC  MA_60_BTC   RSI_BTC  Variation_BTC
0   0.000422   0.000478  0.425431            0.0
1   0.000414   0.000457  0.380366            1.0
2   0.000407   0.000439  0.387446            1.0
```

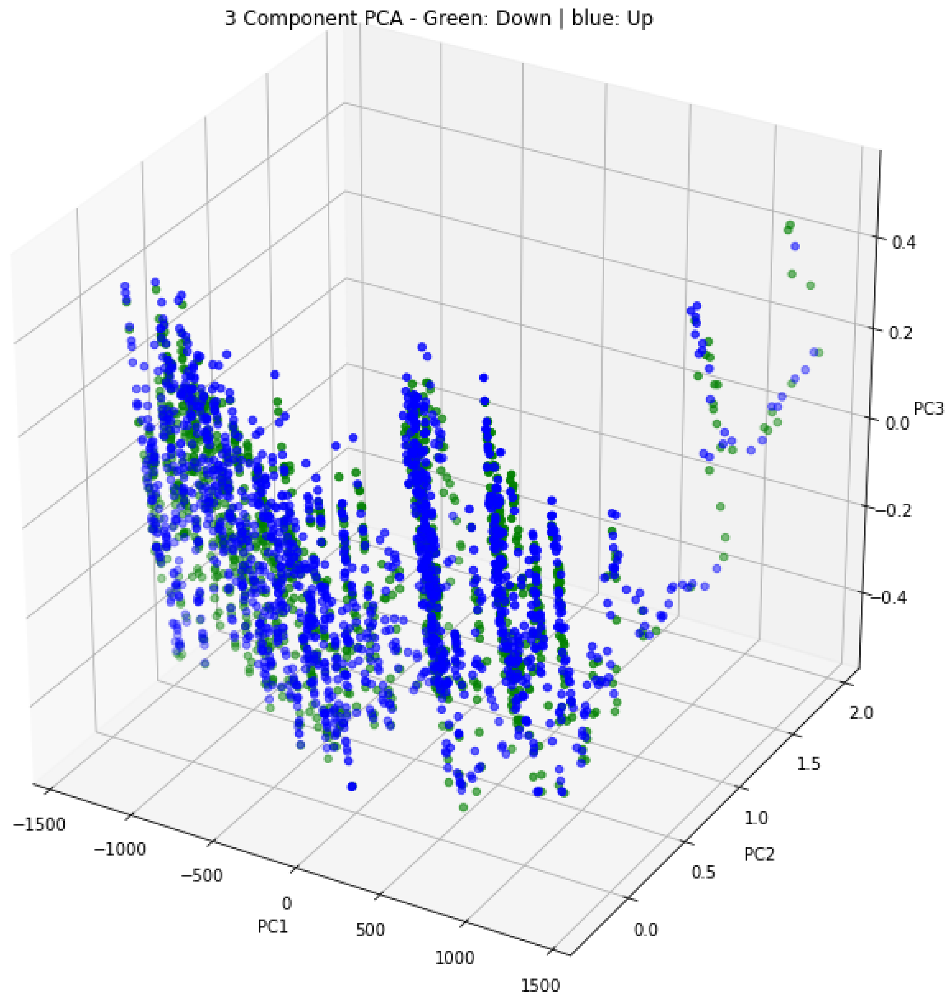
```
3   0.000403   0.000430  0.426791           0.0
4   0.000384   0.000423  0.282513           1.0
```

[5]:
```python
# Correlation with my features for comparison with the TA libairy
sns.heatmap(df_My_Features.corr(), annot=True, linewidths=.5)
plt.show()
```



[6]:
```python
def Get_PCA(df):

    df_pca = df.copy()
    fig = plt.figure(figsize=(12, 12))
    ax = fig.add_subplot(111, projection='3d')

    label = df_pca.pop(df_pca.columns[-1])

    # Run The PCA
    pca = PCA(n_components=3)
    pca.fit(df_pca)

    # Store results of PCA in a data frame
```

```
    result = pd.DataFrame(pca.transform(df_pca), columns=['PCA1', 'PCA2',␣
↪'PCA3'], index=df.index)

    targets = [0,1]
    colors = ['g', 'b']

    for target, color in zip(targets,colors):
        indicesToKeep = label == target
        ax.scatter(result.loc[indicesToKeep, 'PCA1'],
                   result.loc[indicesToKeep, 'PCA2'],
                   result.loc[indicesToKeep, 'PCA3'],
                   c = color,
                   s = 20)

    # label the axes
    ax.set_xlabel("PC1")
    ax.set_ylabel("PC2")
    ax.set_zlabel("PC3")
    plt.title("3 Component PCA - Green: Down | blue: Up")
    plt.show()

    return(df_pca)

df_My_Features_pca = Get_PCA(df_My_Features)
```



[7]:
```
# DF using all the technicals indicators

# df = Pre_Processing("Ethereum")
df = Pre_Processing("Bitcoin")
df.head()
```

```
D:\Programs\conda\lib\site-packages\ta\trend.py:768: RuntimeWarning:

invalid value encountered in double_scalars

D:\Programs\conda\lib\site-packages\ta\trend.py:772: RuntimeWarning:

invalid value encountered in double_scalars
```

[7]:
```
   index  High_BTC   Low_BTC  Open_BTC  Close_BTC  Volume_BTC  Marketcap_BTC  \
0      0  0.001252  0.001231  0.001147   0.001324         0.0       0.000770
1      1  0.001242  0.001232  0.001314   0.001228         0.0       0.000713
2      2  0.001121  0.000759  0.001227   0.000845         0.0       0.000486
3      3  0.000876  0.000481  0.000833   0.000640         0.0       0.000364
4      4  0.000576  0.000244  0.000657   0.000510         0.0       0.000287

   volume_adi_BTC  volume_obv_BTC  volume_cmf_BTC  …  momentum_ao_BTC  \
0        0.000074        0.000271        0.345694  …         0.230926
1        0.000074        0.000271        0.345694  …         0.230926
2        0.000074        0.000271        0.345694  …         0.230926
3        0.000074        0.000271        0.345694  …         0.230926
4        0.000074        0.000271        0.345694  …         0.230926

   momentum_kama_BTC  momentum_roc_BTC  momentum_ppo_BTC  \
0           0.001229           0.20552          0.248522
1           0.001162           0.20552          0.248522
2           0.000873           0.20552          0.248522
3           0.000614           0.20552          0.248522
4           0.000452           0.20552          0.248522

   momentum_ppo_signal_BTC  momentum_ppo_hist_BTC  others_dr_BTC  \
0                 0.218915               0.472307       0.000000
1                 0.218915               0.472307       0.665682
2                 0.218915               0.472307       0.579949
3                 0.218915               0.472307       0.621133
4                 0.218915               0.472307       0.642411

   others_dlr_BTC  others_cr_BTC  Variation_BTC
0        0.565241       0.001324            0.0
1        0.517706       0.001228            0.0
2        0.355572       0.000845            0.0
3        0.436157       0.000640            0.0
4        0.475789       0.000510            1.0

[5 rows x 91 columns]
```

[8]:
```
# Example on another coin
df_demo = Pre_Processing("Ethereum")
df_demo.head()
```

```
D:\Programs\conda\lib\site-packages\ta\trend.py:768: RuntimeWarning:

invalid value encountered in double_scalars

D:\Programs\conda\lib\site-packages\ta\trend.py:772: RuntimeWarning:

invalid value encountered in double_scalars
```

[8]:

```
   index  High_ETH   Low_ETH  Open_ETH  Close_ETH  Volume_ETH  Marketcap_ETH  \
0      0  0.001138  0.000155  0.001206   0.000163    0.000009       0.000059
1      1  0.000195  0.000110  0.000140   0.000136    0.000007       0.000045
2      2  0.000121  0.000114  0.000144   0.000140    0.000005       0.000047
3      3  0.000319  0.000128  0.000141   0.000323    0.000022       0.000144
4      4  0.000396  0.000244  0.000320   0.000399    0.000034       0.000184

   volume_adi_ETH  volume_obv_ETH  volume_cmf_ETH  …  momentum_ao_ETH  \
0        0.000004        0.026348        0.000000  …         0.384313
1        0.000004        0.026348        0.156616  …         0.384313
2        0.000004        0.026348        0.364462  …         0.384313
3        0.000005        0.026349        0.717123  …         0.384313
4        0.000005        0.026351        0.854039  …         0.384313

   momentum_kama_ETH  momentum_roc_ETH  momentum_ppo_ETH  \
0           0.000105          0.247591          0.408479
1           0.000083          0.247591          0.392520
2           0.000080          0.247591          0.364676
3           0.000230          0.247591          0.461176
4           0.000316          0.247591          0.579914

   momentum_ppo_signal_ETH  momentum_ppo_hist_ETH  others_dr_ETH  \
0                 0.439364               0.341845       0.000000
1                 0.435454               0.314344       0.617396
2                 0.425504               0.271865       0.668979
3                 0.441187               0.452145       1.000000
4                 0.482826               0.634688       0.755885

   others_dlr_ETH  others_cr_ETH  Variation_ETH
0        0.573042       0.000163            0.0
1        0.499467       0.000136            1.0
2        0.582708       0.000140            1.0
3        1.000000       0.000323            1.0
4        0.709446       0.000399            1.0

[5 rows x 91 columns]
```

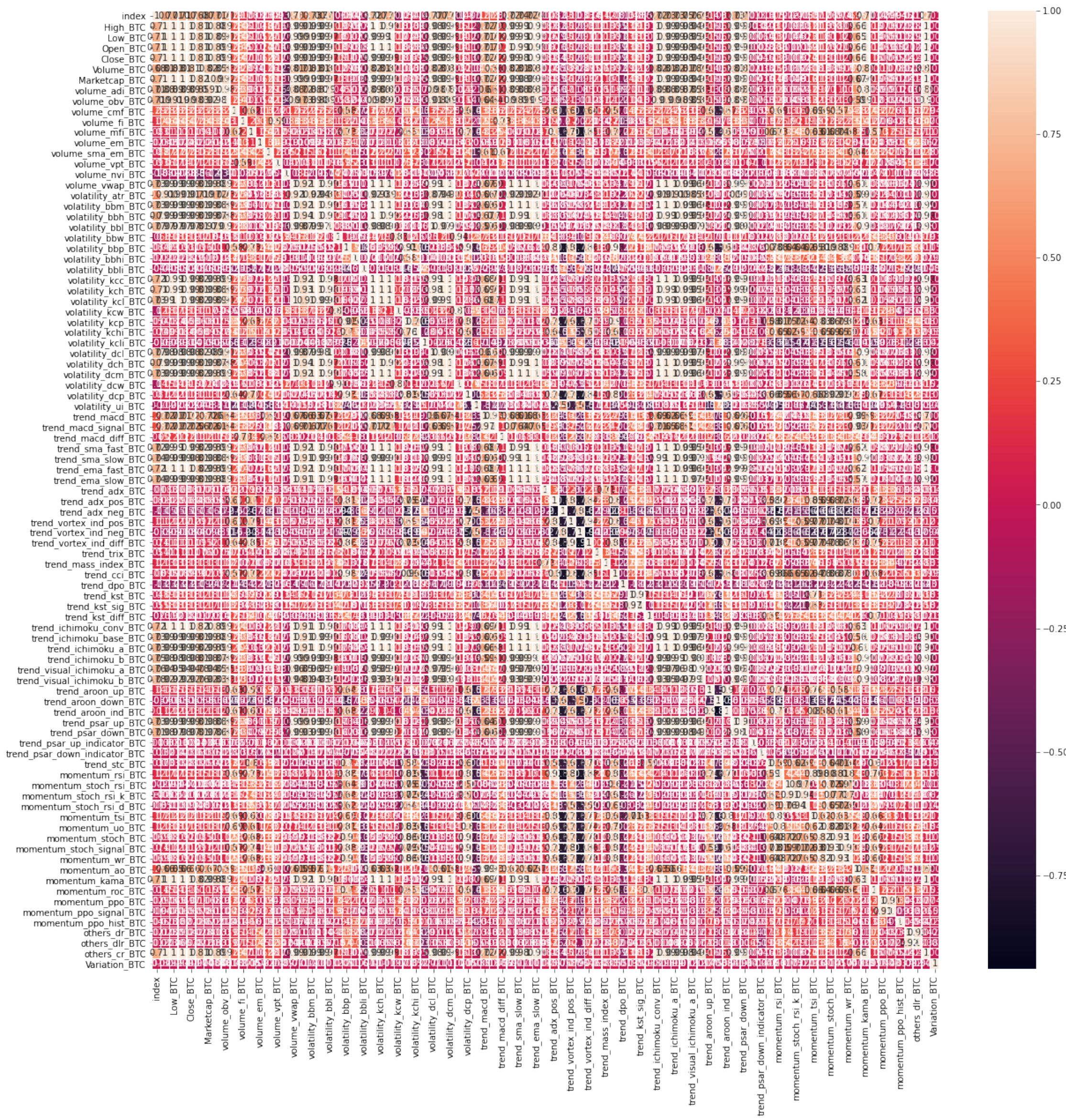
[9]:

```
# df.info(verbose=True)
print(df.dtypes)
```

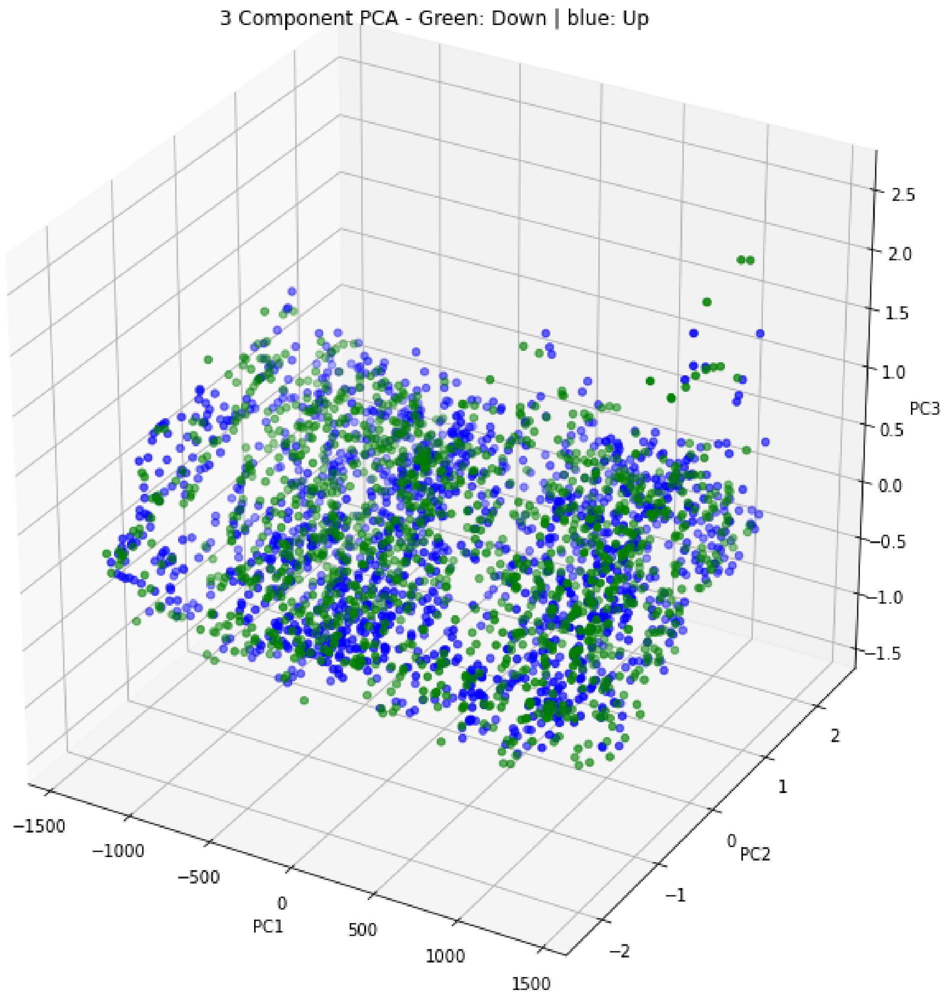
```
index                          int64
High_BTC                     float64
Low_BTC                      float64
Open_BTC                     float64
Close_BTC                    float64
                               …
```

```
momentum_ppo_hist_BTC    float64
others_dr_BTC            float64
others_dlr_BTC           float64
others_cr_BTC            float64
Variation_BTC            float64
Length: 91, dtype: object
```

[10]:
```
plt.figure(figsize = (20,20))
sns.heatmap(df.corr(), annot=True, linewidths=.5)
plt.show()
```



[11]:
```
df_pca = Get_PCA(df)
```



3 Part II: Dense Layer Model

In this part, I recreated a basic Neural Network similar to the one used in the previous assignements. I used 5 Dense layers from Keras with sigmoid activation and a last one to go back to the (None,2) shape of the categorical labels.

However, regardless of the modification I made to the network, I coudn't managed to get it to learn anything. My guess is that, it couldn't create complex enough operations to actually predict something. As a result, the callback stops the training each time because the validation categorical accuracy didn't improve.

[12]:
```
model = tf.keras.models.Sequential([
 Dense(1024, batch_input_shape=(None, df.shape[1]-1),activation="sigmoid"),
 Dense(2048, activation="sigmoid"),
```

```
    Dense(4096, activation="sigmoid"),
    Dense(8192, activation="sigmoid"),
    Dense(2, activation="softmax"),
    ])

sgd = tf.optimizers.SGD(lr=1e-4)
model.compile(optimizer=sgd, loss='categorical_crossentropy',␣
 ↪metrics=['categorical_accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1024)              93184
_________________________________________________________________
dense_1 (Dense)              (None, 2048)              2099200
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              8392704
_________________________________________________________________
dense_3 (Dense)              (None, 8192)              33562624
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 16386
=================================================================
Total params: 44,164,098
Trainable params: 44,164,098
Non-trainable params: 0
_________________________________________________________________
```

[13]:
```
X = df.copy()
Y = X.pop(X.columns[-1])
Y_c = tf.keras.utils.to_categorical(Y)

X_train, X_test, y_train, y_test = train_test_split(X, Y_c, test_size=0.2,␣
 ↪random_state=1, stratify=Y_c)
```

[14]:
```
callback = tf.keras.callbacks.EarlyStopping(monitor='val_categorical_accuracy',␣
 ↪patience=20)
history = model.fit(X_train, y_train, epochs=200, batch_size=64,␣
 ↪validation_data=(X_test, y_test), callbacks=[callback], verbose=2)
```

```
Epoch 1/200
36/36 - 8s - loss: 0.7462 - categorical_accuracy: 0.5420 - val_loss: 0.6897 -
val_categorical_accuracy: 0.5428
Epoch 2/200
36/36 - 7s - loss: 0.6900 - categorical_accuracy: 0.5420 - val_loss: 0.6896 -
val_categorical_accuracy: 0.5428
```

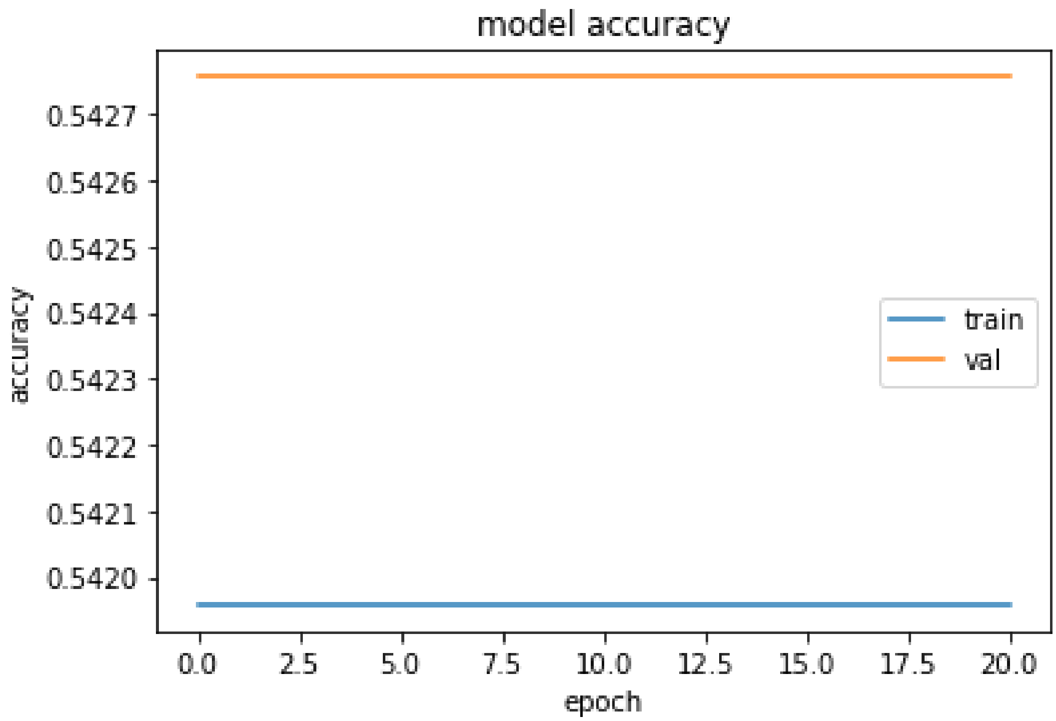
```
Epoch 3/200
36/36 - 7s - loss: 0.6900 - categorical_accuracy: 0.5420 - val_loss: 0.6900 -
val_categorical_accuracy: 0.5428
Epoch 4/200
36/36 - 7s - loss: 0.6910 - categorical_accuracy: 0.5420 - val_loss: 0.6896 -
val_categorical_accuracy: 0.5428
Epoch 5/200
36/36 - 7s - loss: 0.6904 - categorical_accuracy: 0.5420 - val_loss: 0.6896 -
val_categorical_accuracy: 0.5428
Epoch 6/200
36/36 - 7s - loss: 0.6905 - categorical_accuracy: 0.5420 - val_loss: 0.6895 -
val_categorical_accuracy: 0.5428
Epoch 7/200
36/36 - 7s - loss: 0.6902 - categorical_accuracy: 0.5420 - val_loss: 0.6896 -
val_categorical_accuracy: 0.5428
Epoch 8/200
36/36 - 7s - loss: 0.6903 - categorical_accuracy: 0.5420 - val_loss: 0.6897 -
val_categorical_accuracy: 0.5428
Epoch 9/200
36/36 - 7s - loss: 0.6903 - categorical_accuracy: 0.5420 - val_loss: 0.6896 -
val_categorical_accuracy: 0.5428
Epoch 10/200
36/36 - 6s - loss: 0.6903 - categorical_accuracy: 0.5420 - val_loss: 0.6895 -
val_categorical_accuracy: 0.5428
Epoch 11/200
36/36 - 7s - loss: 0.6901 - categorical_accuracy: 0.5420 - val_loss: 0.6901 -
val_categorical_accuracy: 0.5428
Epoch 12/200
36/36 - 7s - loss: 0.6908 - categorical_accuracy: 0.5420 - val_loss: 0.6899 -
val_categorical_accuracy: 0.5428
Epoch 13/200
36/36 - 6s - loss: 0.6909 - categorical_accuracy: 0.5420 - val_loss: 0.6895 -
val_categorical_accuracy: 0.5428
Epoch 14/200
36/36 - 6s - loss: 0.6905 - categorical_accuracy: 0.5420 - val_loss: 0.6898 -
val_categorical_accuracy: 0.5428
Epoch 15/200
36/36 - 6s - loss: 0.6902 - categorical_accuracy: 0.5420 - val_loss: 0.6898 -
val_categorical_accuracy: 0.5428
Epoch 16/200
36/36 - 6s - loss: 0.6904 - categorical_accuracy: 0.5420 - val_loss: 0.6898 -
val_categorical_accuracy: 0.5428
Epoch 17/200
36/36 - 6s - loss: 0.6904 - categorical_accuracy: 0.5420 - val_loss: 0.6896 -
val_categorical_accuracy: 0.5428
Epoch 18/200
36/36 - 7s - loss: 0.6906 - categorical_accuracy: 0.5420 - val_loss: 0.6899 -
val_categorical_accuracy: 0.5428
```

```
Epoch 19/200
36/36 - 7s - loss: 0.6901 - categorical_accuracy: 0.5420 - val_loss: 0.6901 -
val_categorical_accuracy: 0.5428
Epoch 20/200
36/36 - 7s - loss: 0.6903 - categorical_accuracy: 0.5420 - val_loss: 0.6896 -
val_categorical_accuracy: 0.5428
Epoch 21/200
36/36 - 7s - loss: 0.6902 - categorical_accuracy: 0.5420 - val_loss: 0.6896 -
val_categorical_accuracy: 0.5428
```

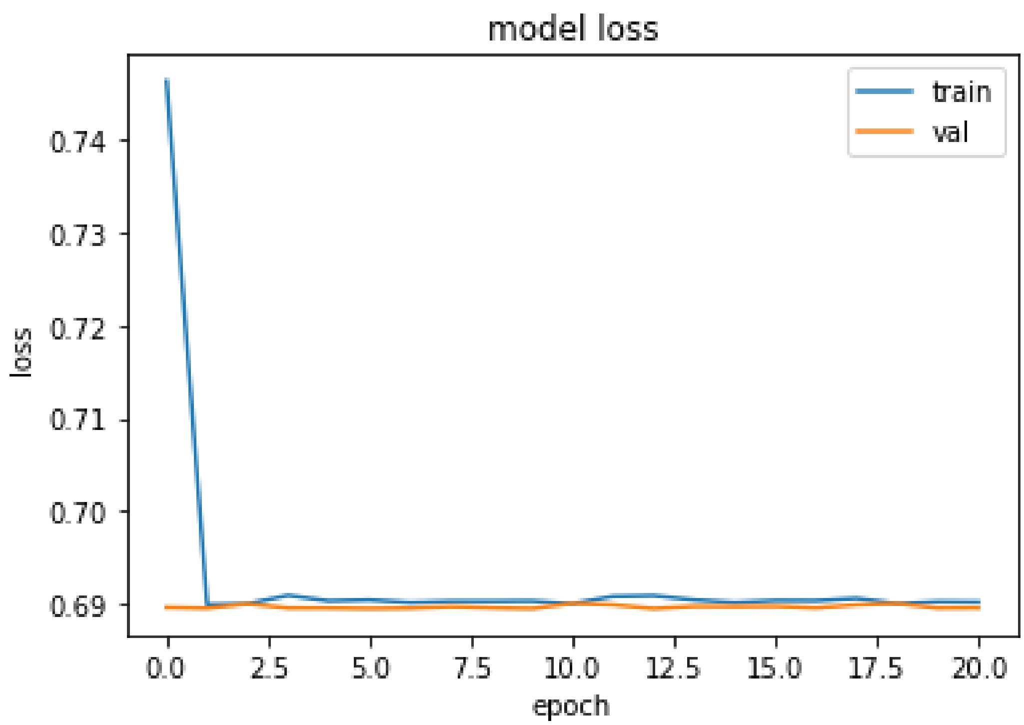
[15]:

```
print(history.history.keys())
plt.plot(history.history['categorical_accuracy'])
plt.plot(history.history['val_categorical_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'])
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'])
plt.show()
```

```
dict_keys(['loss', 'categorical_accuracy', 'val_loss',
'val_categorical_accuracy'])
```



model accuracy
0.5427
0.5426
0.5425
0.5424
0.5423
0.5422
0.5421
0.5420
accuracy
train
val
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
epoch



model loss
0.74
0.73
0.72
0.71
0.70
0.69
loss
train
val
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
epoch

4 Part III: Decision Tree Classifier

In this part, I wanted to see how the basic concept of decision tree seen in class would behave on this complex case.

I noticed that because the entropy is linked to the number of labels and its related features, therefore I equalized the representation of each label during the second run and we see how it affects the classification.

[16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, np.argmin(Y_c, axis=1),␣
↪test_size=0.2, random_state=1)
clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100,␣
↪max_depth = 3, min_samples_leaf = 5)
clf_entropy.fit(X_train, y_train)
y_pred = clf_entropy.predict(X_test)

print("Confusion Matrix:", "\n", confusion_matrix(y_test, y_pred), "\n")
print("Accuracy Validation:", "\n", accuracy_score(y_test,y_pred)*100, "\n")
print("Confusion Matrix:", "\n", classification_report(y_test, y_pred), "\n")
```

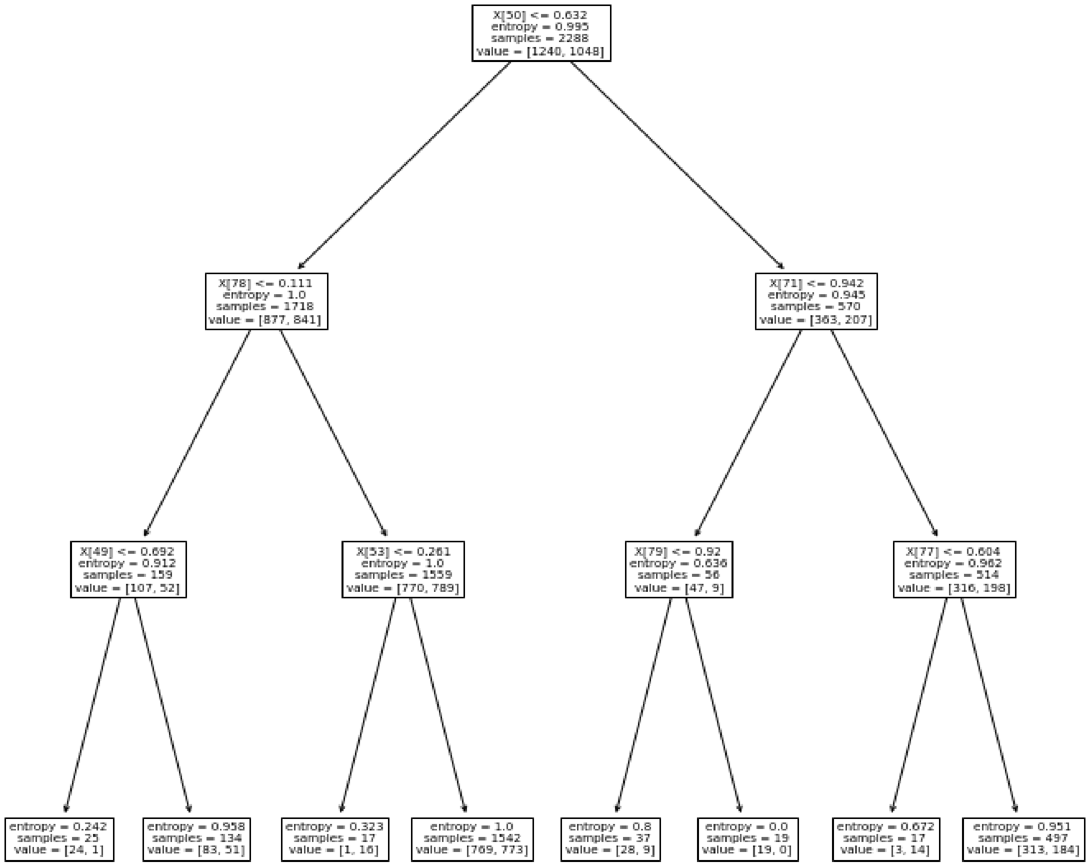
```
Confusion Matrix:
 [[ 73 255]
 [ 49 196]]

Accuracy Validation:
 46.94589877835951

Confusion Matrix:
               precision    recall  f1-score   support

           0       0.60      0.22      0.32       328
           1       0.43      0.80      0.56       245

    accuracy                           0.47       573
   macro avg       0.52      0.51      0.44       573
weighted avg       0.53      0.47      0.43       573
```

[17]:
```
# With equal label representation

X_train, X_test, y_train, y_test = train_test_split(X, np.argmin(Y_c, axis=1),␣
↪test_size=0.2, random_state=1, stratify=Y)
clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100,␣
↪max_depth = 3, min_samples_leaf = 5)
clf_entropy.fit(X_train, y_train)
y_pred = clf_entropy.predict(X_test)
```

```
print("Confusion Matrix:", "\n", confusion_matrix(y_test, y_pred), "\n")
print("Accuracy Validation:", "\n", accuracy_score(y_test,y_pred)*100, "\n")
print("Confusion Matrix:", "\n", classification_report(y_test, y_pred), "\n")
```

```
Confusion Matrix:
 [[108 203]
 [ 86 176]]

Accuracy Validation:
 49.56369982547993

Confusion Matrix:
               precision    recall  f1-score   support

           0       0.56      0.35      0.43       311
           1       0.46      0.67      0.55       262

    accuracy                           0.50       573
   macro avg       0.51      0.51      0.49       573
weighted avg       0.51      0.50      0.48       573
```

[18]:
```
plt.figure(figsize = (12,12))
tree.plot_tree(clf_entropy)
plt.show()
```



5 Part IV: Long & Short Term Memory Cell

Here is the essence of my work. I Implemented a Long & Short Term Memory & CNN Model. To do so, I used the help of the following ressource: https://towardsdatascience.com/the-beginning-of-a-deep-learning-trading-bot-part1-95-accuracy-is-not-enough-c338abc98fc2.

I'm not sure why, but I ran into a lot of troubles when trying to implement a binary cross entropy, so I had to change the label. For this application I'm not having 1 or 0 for the variation but instead I'm directly predicting the closing price.

Because I'm doing a 1D convolution, I have to change the shape of the features and the label into "packages" on which I can train. The hyperparameter that defines this size is the sequence, and I set it to 64. Since we have 89 features, the shapes are now: (None, 64, 89) and for the labels (None,).

Then, training using the following network I managed up 93% of accuracy on the price prediction on the validation set!! In fact in the best epoch, the Mean Average Error was down to 7%, so this network from one day to another could predict the price of the Bitcoin (We can compare the performance on other coins by calling another dataset).

However, even if those results are very promessing, this is not precise enough to be hoping to make an automated process out of this network, and I will detail possible improvements in the conclusion. Please find the network, in the following part.

[19]:
```
# df = Pre_Processing("Litecoin")
# df = Pre_Processing("Monero")
# df = Pre_Processing("Cosmos")
# df = Pre_Processing("Ethereum")
df = Pre_Processing("Bitcoin")


df.pop(df.columns[-1]) # Removing the variation

df.tail()
```

```
D:\Programs\conda\lib\site-packages\ta\trend.py:768: RuntimeWarning:

invalid value encountered in double_scalars

D:\Programs\conda\lib\site-packages\ta\trend.py:772: RuntimeWarning:

invalid value encountered in double_scalars
```

[19]:
```
      index  High_BTC   Low_BTC  Open_BTC  Close_BTC  Volume_BTC  \
2856   2856  0.986316  0.879421  1.000000   0.942013    0.262282
2857   2857  0.929181  0.813297  0.942089   0.848351    0.302314
2858   2858  0.879147  0.847877  0.848642   0.863678    0.181485
2859   2859  0.890456  0.845725  0.863852   0.818239    0.155304
2860   2860  0.829034  0.798267  0.819848   0.805118    1.000000

      Marketcap_BTC  volume_adi_BTC  volume_obv_BTC  volume_cmf_BTC  …  \
2856       0.942082        1.000000        0.965750        0.627135  …
2857       0.848493        0.993881        0.926271        0.565683  …
2858       0.863862        0.997826        0.949971        0.596194  …
2859       0.818479        0.982652        0.929690        0.527725  …
2860       0.805400        0.979007        0.799104        0.498546  …

      momentum_wr_BTC  momentum_ao_BTC  momentum_kama_BTC  momentum_roc_BTC  \
2856         0.714642         1.000000           1.000000          0.303304
2857         0.342109         0.951797           0.999033          0.214552
2858         0.390133         0.874597           0.998523          0.227423
2859         0.138287         0.768105           0.997705          0.205403
```

```
2860        0.135843        0.621920         0.995259         0.182445

      momentum_ppo_BTC  momentum_ppo_signal_BTC  momentum_ppo_hist_BTC  \
2856          0.253215                 0.229611               0.454705
2857          0.278688                 0.233743               0.522317
2858          0.266620                 0.234540               0.481952
2859          0.249737                 0.231668               0.437543
2860          0.417670                 0.264280               0.867024

      others_dr_BTC  others_dlr_BTC  others_cr_BTC
2856       0.651688        0.492673       0.942013
2857       0.622126        0.438036       0.848351
2858       0.705950        0.586990       0.863678
2859       0.655531        0.499598       0.818239
2860       0.681623        0.545607       0.805118

[5 rows x 90 columns]
```

[20]:
```python
# Creating the packages of data

X = df.copy()
Y = X.pop(X.columns[4]) #CLOSE VALUE !!
sequence = 64

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
↪random_state=1)
y_train = np.array(y_train)
y_test = np.array(y_test)

XX = []
YY = []

for i in range(sequence, len(X_train)):
    XX.append(X_train[i-sequence:i])
    YY.append(y_train[i])
X_train, y_train = np.array(XX), np.array(YY)


XX = []
YY = []

for i in range(sequence, len(X_test)):
    XX.append(X_test[i-sequence:i])
    YY.append(y_test[i])
X_test, y_test = np.array(XX), np.array(YY)

print(X_train.shape, y_train.shape)
```

```
print(X_test.shape, y_test.shape)
```

```
(2224, 64, 89) (2224,)
(509, 64, 89) (509,)
```

[21]:
```
def Inception_A(layer_in, c7):
    branch1x1_1 = Conv1D(c7, kernel_size=1, padding="same",␣
↪use_bias=False)(layer_in)
    branch1x1 = BatchNormalization()(branch1x1_1)
    branch1x1 = ReLU()(branch1x1)

    branch5x5_1 = Conv1D(c7, kernel_size=1, padding='same',␣
↪use_bias=False)(layer_in)
    branch5x5 = BatchNormalization()(branch5x5_1)
    branch5x5 = ReLU()(branch5x5)
    branch5x5 = Conv1D(c7, kernel_size=5, padding='same',␣
↪use_bias=False)(branch5x5)
    branch5x5 = BatchNormalization()(branch5x5)
    branch5x5 = ReLU()(branch5x5)

    branch3x3_1 = Conv1D(c7, kernel_size=1, padding='same',␣
↪use_bias=False)(layer_in)
    branch3x3 = BatchNormalization()(branch3x3_1)
    branch3x3 = ReLU()(branch3x3)
    branch3x3 = Conv1D(c7, kernel_size=3, padding='same',␣
↪use_bias=False)(branch3x3)
    branch3x3 = BatchNormalization()(branch3x3)
    branch3x3 = ReLU()(branch3x3)
    branch3x3 = Conv1D(c7, kernel_size=3, padding='same',␣
↪use_bias=False)(branch3x3)
    branch3x3 = BatchNormalization()(branch3x3)
    branch3x3 = ReLU()(branch3x3)

    branch_pool = AveragePooling1D(pool_size=(3), strides=1,␣
↪padding='same')(layer_in)
    branch_pool = Conv1D(c7, kernel_size=1, padding='same',␣
↪use_bias=False)(branch_pool)
    branch_pool = BatchNormalization()(branch_pool)
    branch_pool = ReLU()(branch_pool)
    outputs = Concatenate(axis=-1)([branch1x1, branch5x5, branch3x3,␣
↪branch_pool])
    return outputs


def Inception_B(layer_in, c7):
    branch3x3 = Conv1D(c7, kernel_size=3, padding="same", strides=2,␣
↪use_bias=False)(layer_in)
```

```
    branch3x3 = BatchNormalization()(branch3x3)
    branch3x3 = ReLU()(branch3x3)

    branch3x3dbl = Conv1D(c7, kernel_size=1, padding="same",␣
↪use_bias=False)(layer_in)
    branch3x3dbl = BatchNormalization()(branch3x3dbl)
    branch3x3dbl = ReLU()(branch3x3dbl)
    branch3x3dbl = Conv1D(c7, kernel_size=3, padding="same",␣
↪use_bias=False)(branch3x3dbl)
    branch3x3dbl = BatchNormalization()(branch3x3dbl)
    branch3x3dbl = ReLU()(branch3x3dbl)
    branch3x3dbl = Conv1D(c7, kernel_size=3, padding="same", strides=2,␣
↪use_bias=False)(branch3x3dbl)
    branch3x3dbl = BatchNormalization()(branch3x3dbl)
    branch3x3dbl = ReLU()(branch3x3dbl)

    branch_pool = MaxPooling1D(pool_size=3, strides=2, padding="same")(layer_in)

    outputs = Concatenate(axis=-1)([branch3x3, branch3x3dbl, branch_pool])
    return outputs


def Inception_C(layer_in, c7):
    branch1x1_1 = Conv1D(c7, kernel_size=1, padding="same",␣
↪use_bias=False)(layer_in)
    branch1x1 = BatchNormalization()(branch1x1_1)
    branch1x1 = ReLU()(branch1x1)

    branch7x7_1 = Conv1D(c7, kernel_size=1, padding="same",␣
↪use_bias=False)(layer_in)
    branch7x7 = BatchNormalization()(branch7x7_1)
    branch7x7 = ReLU()(branch7x7)
    branch7x7 = Conv1D(c7, kernel_size=(7), padding="same",␣
↪use_bias=False)(branch7x7)
    branch7x7 = BatchNormalization()(branch7x7)
    branch7x7 = ReLU()(branch7x7)
    branch7x7 = Conv1D(c7, kernel_size=(1), padding="same",␣
↪use_bias=False)(branch7x7)
    branch7x7 = BatchNormalization()(branch7x7)
    branch7x7 = ReLU()(branch7x7)

    branch7x7dbl_1 = Conv1D(c7, kernel_size=1, padding="same",␣
↪use_bias=False)(layer_in)
    branch7x7dbl = BatchNormalization()(branch7x7dbl_1)
    branch7x7dbl = ReLU()(branch7x7dbl)
```

```
    branch7x7dbl = Conv1D(c7, kernel_size=(7), padding="same", use_bias=False)(branch7x7dbl)
    branch7x7dbl = BatchNormalization()(branch7x7dbl)
    branch7x7dbl = ReLU()(branch7x7dbl)
    branch7x7dbl = Conv1D(c7, kernel_size=(1), padding="same", use_bias=False)(branch7x7dbl)
    branch7x7dbl = BatchNormalization()(branch7x7dbl)
    branch7x7dbl = ReLU()(branch7x7dbl)
    branch7x7dbl = Conv1D(c7, kernel_size=(7), padding="same", use_bias=False)(branch7x7dbl)
    branch7x7dbl = BatchNormalization()(branch7x7dbl)
    branch7x7dbl = ReLU()(branch7x7dbl)
    branch7x7dbl = Conv1D(c7, kernel_size=(1), padding="same", use_bias=False)(branch7x7dbl)
    branch7x7dbl = BatchNormalization()(branch7x7dbl)
    branch7x7dbl = ReLU()(branch7x7dbl)

    branch_pool = AveragePooling1D(pool_size=3, strides=1, padding='same')(layer_in)
    branch_pool = Conv1D(c7, kernel_size=1, padding='same', use_bias=False)(branch_pool)
    branch_pool = BatchNormalization()(branch_pool)
    branch_pool = ReLU()(branch_pool)

    outputs = Concatenate(axis=-1)([branch1x1, branch7x7, branch7x7dbl, branch_pool])
    return outputs

def create_model():

    in_seq = Input(shape=(sequence, df.shape[1] - 1))

    x = Inception_A(in_seq, 32)
    x = Inception_A(x, 32)
    x = Inception_B(x, 32)
    x = Inception_B(x, 32)
    x = Inception_C(x, 32)
    x = Inception_C(x, 32)

    x = Bidirectional(LSTM(128, return_sequences=True))(x)
    x = Bidirectional(LSTM(128, return_sequences=True))(x)
    x = Bidirectional(LSTM(64, return_sequences=True))(x)

    avg_pool = GlobalAveragePooling1D()(x)
    max_pool = GlobalMaxPooling1D()(x)
```

```
    conc = concatenate([avg_pool, max_pool])
    conc = Dense(64, activation="relu")(conc)

    out = Dense(1, activation="sigmoid")(conc)

    model = Model(inputs=in_seq, outputs=out)
    model.compile(loss="mse", optimizer="adam", metrics=['mae'])
    return model

model = create_model()
model.summary()
```

```
Model: "model"
__________________________________________________________________________________
________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
================
input_1 (InputLayer)            [(None, 64, 89)]     0

__________________________________________________________________________________
________________
conv1d_3 (Conv1D)               (None, 64, 32)       2848        input_1[0][0]

__________________________________________________________________________________
________________
batch_normalization_3 (BatchNor (None, 64, 32)       128         conv1d_3[0][0]

__________________________________________________________________________________
________________
re_lu_3 (ReLU)                  (None, 64, 32)       0
batch_normalization_3[0][0]

__________________________________________________________________________________
________________
conv1d_1 (Conv1D)               (None, 64, 32)       2848        input_1[0][0]

__________________________________________________________________________________
________________
conv1d_4 (Conv1D)               (None, 64, 32)       3072        re_lu_3[0][0]

__________________________________________________________________________________
________________
batch_normalization_1 (BatchNor (None, 64, 32)       128         conv1d_1[0][0]

__________________________________________________________________________________
________________
batch_normalization_4 (BatchNor (None, 64, 32)       128         conv1d_4[0][0]

__________________________________________________________________________________
________________
re_lu_1 (ReLU)                  (None, 64, 32)       0
batch_normalization_1[0][0]

__________________________________________________________________________________
________________
re_lu_4 (ReLU)                  (None, 64, 32)       0
```

```
batch_normalization_4[0][0]
________________________________________________________________________________
__________________
average_pooling1d (AveragePooli (None, 64, 89)       0           input_1[0][0]
________________________________________________________________________________
__________________
conv1d (Conv1D)                 (None, 64, 32)       2848        input_1[0][0]
________________________________________________________________________________
__________________
conv1d_2 (Conv1D)               (None, 64, 32)       5120        re_lu_1[0][0]
________________________________________________________________________________
__________________
conv1d_5 (Conv1D)               (None, 64, 32)       3072        re_lu_4[0][0]
________________________________________________________________________________
__________________
conv1d_6 (Conv1D)               (None, 64, 32)       2848
average_pooling1d[0][0]
________________________________________________________________________________
__________________
batch_normalization (BatchNorma (None, 64, 32)       128         conv1d[0][0]
________________________________________________________________________________
__________________
batch_normalization_2 (BatchNor (None, 64, 32)       128         conv1d_2[0][0]
________________________________________________________________________________
__________________
batch_normalization_5 (BatchNor (None, 64, 32)       128         conv1d_5[0][0]
________________________________________________________________________________
__________________
batch_normalization_6 (BatchNor (None, 64, 32)       128         conv1d_6[0][0]
________________________________________________________________________________
__________________
re_lu (ReLU)                    (None, 64, 32)       0
batch_normalization[0][0]
________________________________________________________________________________
__________________
re_lu_2 (ReLU)                  (None, 64, 32)       0
batch_normalization_2[0][0]
________________________________________________________________________________
__________________
re_lu_5 (ReLU)                  (None, 64, 32)       0
batch_normalization_5[0][0]
________________________________________________________________________________
__________________
re_lu_6 (ReLU)                  (None, 64, 32)       0
batch_normalization_6[0][0]
________________________________________________________________________________
__________________
concatenate (Concatenate)       (None, 64, 128)      0           re_lu[0][0]
```

```
                                                                  re_lu_2[0][0]
                                                                  re_lu_5[0][0]
                                                                  re_lu_6[0][0]
________________________________________________________________________________
__________________
conv1d_10 (Conv1D)              (None, 64, 32)       4096
concatenate[0][0]
________________________________________________________________________________
__________________
batch_normalization_10 (BatchNo (None, 64, 32)       128         conv1d_10[0][0]
________________________________________________________________________________
__________________
re_lu_10 (ReLU)                 (None, 64, 32)       0
batch_normalization_10[0][0]
________________________________________________________________________________
__________________
conv1d_8 (Conv1D)               (None, 64, 32)       4096
concatenate[0][0]
________________________________________________________________________________
__________________
conv1d_11 (Conv1D)              (None, 64, 32)       3072        re_lu_10[0][0]
________________________________________________________________________________
__________________
batch_normalization_8 (BatchNor (None, 64, 32)       128         conv1d_8[0][0]
________________________________________________________________________________
__________________
batch_normalization_11 (BatchNo (None, 64, 32)       128         conv1d_11[0][0]
________________________________________________________________________________
__________________
re_lu_8 (ReLU)                  (None, 64, 32)       0
batch_normalization_8[0][0]
________________________________________________________________________________
__________________
re_lu_11 (ReLU)                 (None, 64, 32)       0
batch_normalization_11[0][0]
________________________________________________________________________________
__________________
average_pooling1d_1 (AveragePoo (None, 64, 128)      0
concatenate[0][0]
________________________________________________________________________________
__________________
conv1d_7 (Conv1D)               (None, 64, 32)       4096
concatenate[0][0]
________________________________________________________________________________
__________________
conv1d_9 (Conv1D)               (None, 64, 32)       5120        re_lu_8[0][0]
________________________________________________________________________________
__________________
```

```
conv1d_12 (Conv1D)              (None, 64, 32)       3072        re_lu_11[0][0]
--------------------------------------------------------------------------------
------------------
conv1d_13 (Conv1D)              (None, 64, 32)       4096
average_pooling1d_1[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_7 (BatchNor (None, 64, 32)       128         conv1d_7[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_9 (BatchNor (None, 64, 32)       128         conv1d_9[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_12 (BatchNo (None, 64, 32)       128         conv1d_12[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_13 (BatchNo (None, 64, 32)       128         conv1d_13[0][0]
--------------------------------------------------------------------------------
------------------
re_lu_7 (ReLU)                  (None, 64, 32)       0
batch_normalization_7[0][0]
--------------------------------------------------------------------------------
------------------
re_lu_9 (ReLU)                  (None, 64, 32)       0
batch_normalization_9[0][0]
--------------------------------------------------------------------------------
------------------
re_lu_12 (ReLU)                 (None, 64, 32)       0
batch_normalization_12[0][0]
--------------------------------------------------------------------------------
------------------
re_lu_13 (ReLU)                 (None, 64, 32)       0
batch_normalization_13[0][0]
--------------------------------------------------------------------------------
------------------
concatenate_1 (Concatenate)     (None, 64, 128)      0           re_lu_7[0][0]
                                                                 re_lu_9[0][0]
                                                                 re_lu_12[0][0]
                                                                 re_lu_13[0][0]
--------------------------------------------------------------------------------
------------------
conv1d_15 (Conv1D)              (None, 64, 32)       4096
concatenate_1[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_15 (BatchNo (None, 64, 32)       128         conv1d_15[0][0]
--------------------------------------------------------------------------------
------------------
```

```
re_lu_15 (ReLU)                (None, 64, 32)       0
batch_normalization_15[0][0]

--------------------------------------------------------------------------------
------------------
conv1d_16 (Conv1D)             (None, 64, 32)       3072        re_lu_15[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_16 (BatchNo (None, 64, 32)      128         conv1d_16[0][0]

--------------------------------------------------------------------------------
------------------
re_lu_16 (ReLU)                (None, 64, 32)       0
batch_normalization_16[0][0]

--------------------------------------------------------------------------------
------------------
conv1d_14 (Conv1D)             (None, 32, 32)       12288
concatenate_1[0][0]

--------------------------------------------------------------------------------
------------------
conv1d_17 (Conv1D)             (None, 32, 32)       3072        re_lu_16[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_14 (BatchNo (None, 32, 32)      128         conv1d_14[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_17 (BatchNo (None, 32, 32)      128         conv1d_17[0][0]

--------------------------------------------------------------------------------
------------------
re_lu_14 (ReLU)                (None, 32, 32)       0
batch_normalization_14[0][0]

--------------------------------------------------------------------------------
------------------
re_lu_17 (ReLU)                (None, 32, 32)       0
batch_normalization_17[0][0]

--------------------------------------------------------------------------------
------------------
max_pooling1d (MaxPooling1D)   (None, 32, 128)      0
concatenate_1[0][0]

--------------------------------------------------------------------------------
------------------
concatenate_2 (Concatenate)    (None, 32, 192)      0           re_lu_14[0][0]
                                                                re_lu_17[0][0]

max_pooling1d[0][0]

--------------------------------------------------------------------------------
------------------
conv1d_19 (Conv1D)             (None, 32, 32)       6144
concatenate_2[0][0]

--------------------------------------------------------------------------------
------------------
```

```
batch_normalization_19 (BatchNo (None, 32, 32)       128         conv1d_19[0][0]
________________________________________________________________________________
__________________
re_lu_19 (ReLU)                 (None, 32, 32)       0
batch_normalization_19[0][0]
________________________________________________________________________________
__________________
conv1d_20 (Conv1D)              (None, 32, 32)       3072        re_lu_19[0][0]
________________________________________________________________________________
__________________
batch_normalization_20 (BatchNo (None, 32, 32)       128         conv1d_20[0][0]
________________________________________________________________________________
__________________
re_lu_20 (ReLU)                 (None, 32, 32)       0
batch_normalization_20[0][0]
________________________________________________________________________________
__________________
conv1d_18 (Conv1D)              (None, 16, 32)       18432
concatenate_2[0][0]
________________________________________________________________________________
__________________
conv1d_21 (Conv1D)              (None, 16, 32)       3072        re_lu_20[0][0]
________________________________________________________________________________
__________________
batch_normalization_18 (BatchNo (None, 16, 32)       128         conv1d_18[0][0]
________________________________________________________________________________
__________________
batch_normalization_21 (BatchNo (None, 16, 32)       128         conv1d_21[0][0]
________________________________________________________________________________
__________________
re_lu_18 (ReLU)                 (None, 16, 32)       0
batch_normalization_18[0][0]
________________________________________________________________________________
__________________
re_lu_21 (ReLU)                 (None, 16, 32)       0
batch_normalization_21[0][0]
________________________________________________________________________________
__________________
max_pooling1d_1 (MaxPooling1D)  (None, 16, 192)      0
concatenate_2[0][0]
________________________________________________________________________________
__________________
concatenate_3 (Concatenate)     (None, 16, 256)      0           re_lu_18[0][0]
                                                                 re_lu_21[0][0]

max_pooling1d_1[0][0]
________________________________________________________________________________
__________________
conv1d_26 (Conv1D)              (None, 16, 32)       8192
```

```
concatenate_3[0][0]
__________________________________________________________________________________________________
batch_normalization_26 (BatchNo (None, 16, 32)       128         conv1d_26[0][0]
__________________________________________________________________________________________________
re_lu_26 (ReLU)                 (None, 16, 32)       0
batch_normalization_26[0][0]
__________________________________________________________________________________________________
conv1d_27 (Conv1D)              (None, 16, 32)       7168        re_lu_26[0][0]
__________________________________________________________________________________________________
batch_normalization_27 (BatchNo (None, 16, 32)       128         conv1d_27[0][0]
__________________________________________________________________________________________________
re_lu_27 (ReLU)                 (None, 16, 32)       0
batch_normalization_27[0][0]
__________________________________________________________________________________________________
conv1d_23 (Conv1D)              (None, 16, 32)       8192
concatenate_3[0][0]
__________________________________________________________________________________________________
conv1d_28 (Conv1D)              (None, 16, 32)       1024        re_lu_27[0][0]
__________________________________________________________________________________________________
batch_normalization_23 (BatchNo (None, 16, 32)       128         conv1d_23[0][0]
__________________________________________________________________________________________________
batch_normalization_28 (BatchNo (None, 16, 32)       128         conv1d_28[0][0]
__________________________________________________________________________________________________
re_lu_23 (ReLU)                 (None, 16, 32)       0
batch_normalization_23[0][0]
__________________________________________________________________________________________________
re_lu_28 (ReLU)                 (None, 16, 32)       0
batch_normalization_28[0][0]
__________________________________________________________________________________________________
conv1d_24 (Conv1D)              (None, 16, 32)       7168        re_lu_23[0][0]
__________________________________________________________________________________________________
conv1d_29 (Conv1D)              (None, 16, 32)       7168        re_lu_28[0][0]
__________________________________________________________________________________________________
batch_normalization_24 (BatchNo (None, 16, 32)       128         conv1d_24[0][0]
```

```
--------------------------------------------------------------------------------
------------------
batch_normalization_29 (BatchNo (None, 16, 32)       128         conv1d_29[0][0]
--------------------------------------------------------------------------------
------------------
re_lu_24 (ReLU)                 (None, 16, 32)       0
batch_normalization_24[0][0]
--------------------------------------------------------------------------------
------------------
re_lu_29 (ReLU)                 (None, 16, 32)       0
batch_normalization_29[0][0]
--------------------------------------------------------------------------------
------------------
average_pooling1d_2 (AveragePoo (None, 16, 256)      0
concatenate_3[0][0]
--------------------------------------------------------------------------------
------------------
conv1d_22 (Conv1D)              (None, 16, 32)       8192
concatenate_3[0][0]
--------------------------------------------------------------------------------
------------------
conv1d_25 (Conv1D)              (None, 16, 32)       1024        re_lu_24[0][0]
--------------------------------------------------------------------------------
------------------
conv1d_30 (Conv1D)              (None, 16, 32)       1024        re_lu_29[0][0]
--------------------------------------------------------------------------------
------------------
conv1d_31 (Conv1D)              (None, 16, 32)       8192
average_pooling1d_2[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_22 (BatchNo (None, 16, 32)       128         conv1d_22[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_25 (BatchNo (None, 16, 32)       128         conv1d_25[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_30 (BatchNo (None, 16, 32)       128         conv1d_30[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_31 (BatchNo (None, 16, 32)       128         conv1d_31[0][0]
--------------------------------------------------------------------------------
------------------
re_lu_22 (ReLU)                 (None, 16, 32)       0
batch_normalization_22[0][0]
--------------------------------------------------------------------------------
------------------
re_lu_25 (ReLU)                 (None, 16, 32)       0
```

```
batch_normalization_25[0][0]
__________________________________________________________________________________________________
re_lu_30 (ReLU)                 (None, 16, 32)       0
batch_normalization_30[0][0]
__________________________________________________________________________________________________
re_lu_31 (ReLU)                 (None, 16, 32)       0
batch_normalization_31[0][0]
__________________________________________________________________________________________________
concatenate_4 (Concatenate)     (None, 16, 128)      0           re_lu_22[0][0]
                                                                 re_lu_25[0][0]
                                                                 re_lu_30[0][0]
                                                                 re_lu_31[0][0]
__________________________________________________________________________________________________
conv1d_36 (Conv1D)              (None, 16, 32)       4096
concatenate_4[0][0]
__________________________________________________________________________________________________
batch_normalization_36 (BatchNo (None, 16, 32)       128         conv1d_36[0][0]
__________________________________________________________________________________________________
re_lu_36 (ReLU)                 (None, 16, 32)       0
batch_normalization_36[0][0]
__________________________________________________________________________________________________
conv1d_37 (Conv1D)              (None, 16, 32)       7168        re_lu_36[0][0]
__________________________________________________________________________________________________
batch_normalization_37 (BatchNo (None, 16, 32)       128         conv1d_37[0][0]
__________________________________________________________________________________________________
re_lu_37 (ReLU)                 (None, 16, 32)       0
batch_normalization_37[0][0]
__________________________________________________________________________________________________
conv1d_33 (Conv1D)              (None, 16, 32)       4096
concatenate_4[0][0]
__________________________________________________________________________________________________
conv1d_38 (Conv1D)              (None, 16, 32)       1024        re_lu_37[0][0]
__________________________________________________________________________________________________
batch_normalization_33 (BatchNo (None, 16, 32)       128         conv1d_33[0][0]
__________________________________________________________________________________________________
```

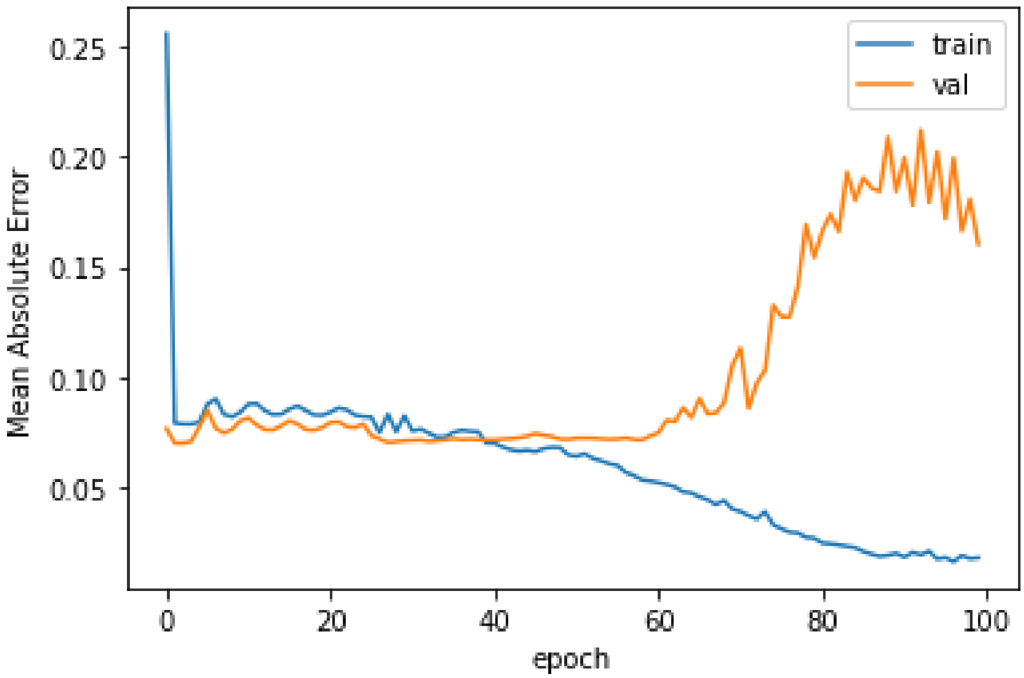
```
batch_normalization_38 (BatchNo (None, 16, 32)       128         conv1d_38[0][0]
__________________________________________________________________________________
__________________
re_lu_33 (ReLU)                 (None, 16, 32)       0
batch_normalization_33[0][0]
__________________________________________________________________________________
__________________
re_lu_38 (ReLU)                 (None, 16, 32)       0
batch_normalization_38[0][0]
__________________________________________________________________________________
__________________
conv1d_34 (Conv1D)              (None, 16, 32)       7168        re_lu_33[0][0]
__________________________________________________________________________________
__________________
conv1d_39 (Conv1D)              (None, 16, 32)       7168        re_lu_38[0][0]
__________________________________________________________________________________
__________________
batch_normalization_34 (BatchNo (None, 16, 32)       128         conv1d_34[0][0]
__________________________________________________________________________________
__________________
batch_normalization_39 (BatchNo (None, 16, 32)       128         conv1d_39[0][0]
__________________________________________________________________________________
__________________
re_lu_34 (ReLU)                 (None, 16, 32)       0
batch_normalization_34[0][0]
__________________________________________________________________________________
__________________
re_lu_39 (ReLU)                 (None, 16, 32)       0
batch_normalization_39[0][0]
__________________________________________________________________________________
__________________
average_pooling1d_3 (AveragePoo (None, 16, 128)      0
concatenate_4[0][0]
__________________________________________________________________________________
__________________
conv1d_32 (Conv1D)              (None, 16, 32)       4096
concatenate_4[0][0]
__________________________________________________________________________________
__________________
conv1d_35 (Conv1D)              (None, 16, 32)       1024        re_lu_34[0][0]
__________________________________________________________________________________
__________________
conv1d_40 (Conv1D)              (None, 16, 32)       1024        re_lu_39[0][0]
__________________________________________________________________________________
__________________
conv1d_41 (Conv1D)              (None, 16, 32)       4096
average_pooling1d_3[0][0]
__________________________________________________________________________________
```

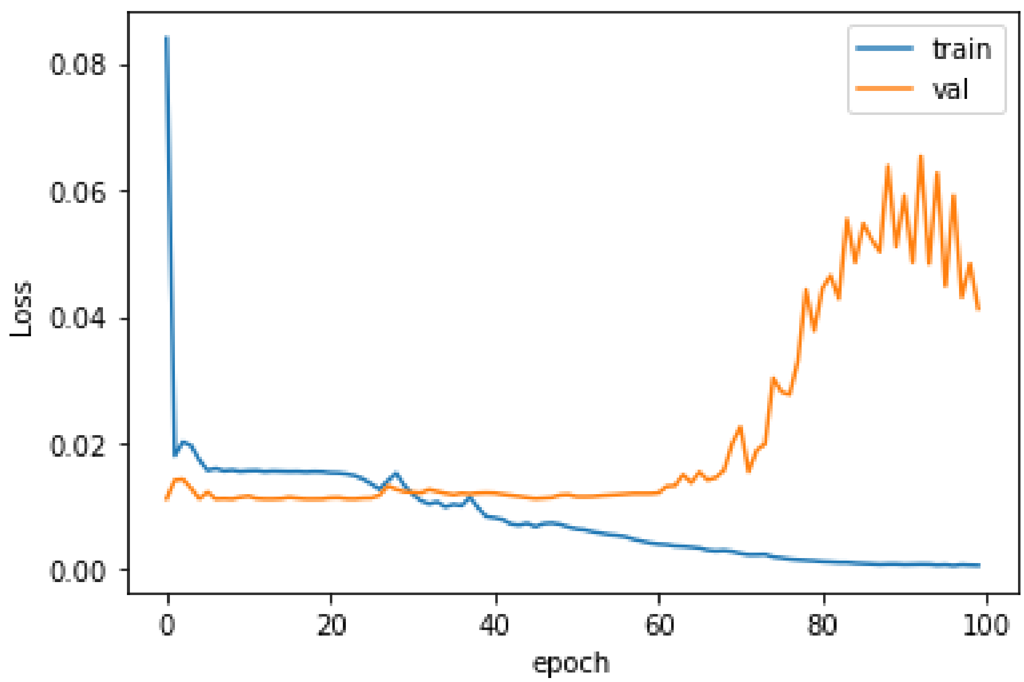
```
------------------
batch_normalization_32 (BatchNo (None, 16, 32)       128         conv1d_32[0][0]
________________________________________________________________________________
------------------
batch_normalization_35 (BatchNo (None, 16, 32)       128         conv1d_35[0][0]
________________________________________________________________________________
------------------
batch_normalization_40 (BatchNo (None, 16, 32)       128         conv1d_40[0][0]
________________________________________________________________________________
------------------
batch_normalization_41 (BatchNo (None, 16, 32)       128         conv1d_41[0][0]
________________________________________________________________________________
------------------
re_lu_32 (ReLU)                 (None, 16, 32)       0
batch_normalization_32[0][0]
________________________________________________________________________________
------------------
re_lu_35 (ReLU)                 (None, 16, 32)       0
batch_normalization_35[0][0]
________________________________________________________________________________
------------------
re_lu_40 (ReLU)                 (None, 16, 32)       0
batch_normalization_40[0][0]
________________________________________________________________________________
------------------
re_lu_41 (ReLU)                 (None, 16, 32)       0
batch_normalization_41[0][0]
________________________________________________________________________________
------------------
concatenate_5 (Concatenate)     (None, 16, 128)      0           re_lu_32[0][0]
                                                                 re_lu_35[0][0]
                                                                 re_lu_40[0][0]
                                                                 re_lu_41[0][0]
________________________________________________________________________________
------------------
bidirectional (Bidirectional)   (None, 16, 256)      263168
concatenate_5[0][0]
________________________________________________________________________________
------------------
bidirectional_1 (Bidirectional) (None, 16, 256)      394240
bidirectional[0][0]
________________________________________________________________________________
------------------
bidirectional_2 (Bidirectional) (None, 16, 128)      164352
bidirectional_1[0][0]
________________________________________________________________________________
------------------
global_average_pooling1d (Globa (None, 128)          0
```

```
bidirectional_2[0][0]

--------------------------------------------------------------------------------
------------------
global_max_pooling1d (GlobalMax (None, 128)          0
bidirectional_2[0][0]

--------------------------------------------------------------------------------
------------------
concatenate_6 (Concatenate)     (None, 256)          0
global_average_pooling1d[0][0]
global_max_pooling1d[0][0]

--------------------------------------------------------------------------------
------------------
dense_5 (Dense)                 (None, 64)           16448
concatenate_6[0][0]

--------------------------------------------------------------------------------
------------------
dense_6 (Dense)                 (None, 1)            65          dense_5[0][0]
================================================================================
==================
Total params: 1,045,505
Trainable params: 1,042,817
Non-trainable params: 2,688

--------------------------------------------------------------------------------
------------------
```

[22]:
```
history = model.fit(X_train, y_train, epochs=100, batch_size=1024,␣
 ↪validation_data=(X_test, y_test), verbose=0)
```

[23]:
```
#Calculate predication for training, validation and test data
history.history.keys()

plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.ylabel('Mean Absolute Error')
plt.xlabel('epoch')
plt.legend(['train', 'val'])
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.ylabel('Loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'])
plt.show()
```





[24]:
```
best_epoch = np.argmin(history.history["val_mae"])
print("The minimum MAE is:",history.history["val_mae"][best_epoch]*100,"%")
```

```
The minimum MAE is: 6.997121870517731 %
```

6 Conclusion - Avenues for improvement:

There are several take away from this work, but first I have to admit that I'm very impressed by the possibilities and the accuracy of the networks out there.

However, predicting the closing value of an asset with a 7

- Reinforced Learning: From the research I made, RL is the prominent technology, because when setting a "gym" and a proper system rewards, the network can learn in a simulation of the real world taking multiples parameters such as the fee the trader as to pay for each transaction. In this said vertical environment, we can see how the portofolio varies and reward the network accordingly. For this project, I started to implement such system but this was too long, however I will do one for myself to obtain a viable trading bot.

- Categorical label UP/DOWN: In order to be profitable, it is not necessary to predict the price, but predicting the direction of the movement might be better since it will be easier for the network. We could imagine a system where the network takes a position as a proportion of the confidence in the prediction, if the confidence justify the various fees.

- Max draw down: We need to quantify an important metric, the "Draw Down", it is the maximum amount of losses a trading network can assume in the event of a catastrophic event or mistake.

- Add New Features: Adding more features to the Dataset can improve the network performances. In the cryptocurrencies world we notice that since the Bitcoin is the prominent asset, it impacts the over values when it varies. Therefore, it can be interesting to concatenate the dataset of the most capitalized currencies to have a 360 degrees visions on the environment. In addition, in order for the network to be able to apprehend real world events, we could be parsing twitter or any other Finance related journal so that we could take in account when those are positively -or negatively- speaking of those.

Thank you for reading me, sorry for exceeding the page cap, I really wanted to dig the subject and I will continue later as a personnal project.