International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

Deep Learning for Financial Time Series using Long Short-Term Memory Model

Aumkar Wagle

In this paper, we will be using the LSTM model to predict the next day's updtrend for financial time series.

The idea here is to analyze whether LSTM is able to predict the sign of daily returns based on adjusted close prices. Thus, it is a classification problem.

I begin with the problem statement of this paper i.e. the use of LSTM model for time series prediction. The data has been cleaned to remove any null values and use the linear interpolation method in case there are any such values.

The pandas TA library has been used in order to generate features. Importing all the stategies present in the library, I have used the BorutaPy method of feature selection as it is a robust method that provides us with the relevant feature ranking that can be used for our model.

Next I transform the features that have been selected. By using MinMaxScaler estimator fromsklearn.preprocessing. With MinMaxScaler we scale our data such that each feature lies between zero and one (mapping 0,1).

Post this, we jump into the model building exercise. I have built three different LSTM models. Each of them has different number of layers with different number of dropout layers.

The reason for building these three types of models is to test the different depths our models and how they are working on the given dataset. There is a shallow model, a model that has a few layers, and a model that has a relatively larger number of layers. Once we have fitted each model on the data and then tested it out, I have built a classification report, confusion matrix and ROC curve plot to compare the 3 models.

Post evaluating how each of the models work, I have chosen two of the best models (as they had similar evaluations) for the hyperparameter tuning process. The reason for choosing two models is as i mentioned, because they had similar accuracy results, so I was intrigued to see how they would perform after the tuning process, given that the layers are different for both models, yet they had similar predictions.

Lastly, on the best model from the above two, I have adopted a simple trading strategy and backtested the same on our data post which I present the concluding remarks.

Problem Statement

The time series that has been used in this particular paper is that of India's Nifty 50 index which is an index of the top 50 stocks of the Indian economy.

Daily data worth 20 years has been used for this project given that LSTM is a deep learning algorithm which usually requires a lot of data to make a sound prediction. While the prediction signal is not of the utmost importance in this project, but rather the model building, optimization, tuning, etc. is of more importance, 20 years worth data was still chosen to fit the requirement of having more data than having less. Additionally, having 20 years worth of data also allows for seeing different regime changes encorporated in the time series due to various economic events such as recessions, crises, etc being encapsulated by the movement in the market. This would help to not have only one type of trend (i.e. up or down) but rather a more balanced set of up moves and down moves which would allow the algorithm to not form a bias in it's prediction. Yet, we would still see that the data isnt perfectly balanced but that has been taken care of as can be seen later on in the code.

First we import all the relevant libraries that are necessary in order to make our analysis in this project.

```
import os, random
import pandas as pd
import numpy as np
import datetime as dt
import pandas_ta as ta
from pathlib import Path

# import boruta
from boruta import BorutaPy

# warnings
import warnings
warnings.filterwarnings('ignore')

# plotting & outputs
from pprint import pprint
import matplotlib.pyplot as plt
plt.style.use('seaborn')

# sklearn imports
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler,
MinMaxScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.model_selection import TimeSeriesSplit,
cross_val_score
from sklearn.model_selection import GridSearchCV,
RandomizedSearchCV
from sklearn.base import BaseEstimator,
TransformerMixin

# metrics
```

```
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import plot_confusion_matrix, auc, roc_curve, plot_roc_curve

# import classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, StackingClassifier
from sklearn.neighbors import KNeighborsClassifier

# metrics
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score
from sklearn.metrics import classification_report, confusion_matrix
import scikitplot
from scikitplot.metrics import plot_confusion_matrix, plot_roc

# tensorflow
import tensorflow as tf
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import Sequential, Model, load_model
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator

from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.metrics import BinaryAccuracy, Accuracy, AUC, Precision, Recall
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, TensorBoard
from tensorflow.keras.layers import Dropout, Dense, Flatten
from tensorflow.keras.layers import LSTM, BatchNormalization

# kerastuner
import keras_tuner as kt
from kerastuner import HyperParameter, HyperParameters
from kerastuner.tuners import RandomSearch, BayesianOptimization, Hyperband
```

Here we are creating a function to define the seed and class weights.

```
# define seed
def set_seeds(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)

# class weight function
def cwts(dfs):
    c0, c1 = np.bincount(dfs)
    w0=(1/c0)*(len(dfs))/2
    w1=(1/c1)*(len(dfs))/2
    return {0: w0, 1: w1}
```

I am using a work computer to run this analysis and the Yahoo Finance API is pinged quite often so it isnt allowing me to download the data directly. Hence, I have downloaded the data manually onto my computer and am using the pandas read function to read the data that is stored in the given directory.

```
#reading and plotting the downloaded data
df = pd.read_csv('Nifty_50.csv', index_col=0, parse_dates=True)[['Open', 'High', 'Low', 'Close','Adj Close','Volume']]
df.shape
plt.plot(df['Adj Close']);
df.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
13-01-2014	6189.549805	6288.200195	6189.549805	6272.75	6272.75	135000
14-01-2014	6260.25	6280.350098	6234.149902	6241.850098	6241.850098	110200
15-01-2014	6265.950195	6325.200195	6265.299805	6320.899902	6320.899902	145900
16-01-2014	6341.350098	6346.5	6299.850098	6318.899902	6318.899902	153400
17-01-2014	6306.25	6327.100098	6246.350098	6261.649902	6261.649902	167700

Above we can see that we have plotted the data to see what the time series looks like. As we can see, we are capturing different regimes in our data going back to 2014. There is a pick up in the levels post the Global Financial Crisis and then we can see a drop aroun the 2020 period when the COVID-19 pandemic occured. Post this, we again see an uptrend in the levels of the index.

Exploratory Data Analysis

df.describe()

	Open	High	Low	Close	Adj Close	Volume
count	2455	2455	2455	2455	2455	2.46E+03
mean	11965.44668	12020.67994	11889.34703	11956.82612	11956.82612	3.07E+05
std	3998.931175	4011.896283	3979.714669	3997.831375	3997.831375	1.98E+05
min	5947.600098	6017.799805	5933.299805	6000.899902	6000.899902	0.00E+00
25%	8544.475098	8595.375	8504.050293	8541.300293	8541.300293	1.76E+05
50%	10804.84961	10836.84961	10736.09961	10792.5	10792.5	2.43E+05
75%	15784.15039	15855.17481	15705.1001	15773.5752	15773.5752	3.69E+05
max	21773.55078	21928.25	21715.15039	21894.55078	21894.55078	1.81E+06

Above we can see that there isnt anything out of the ordinary in terms of the distributional aspects of our dataset that we are using. Since we are using data over a relatively larger period of time, any excessive outliers will be smoothened out with regards to any eye-catching statistical outputs that they would otherwise bring about.

Data Cleaning

I observed that the data needed to be cleaned in order for me to have the pandas TA library import all the strategies correctly. This is because there were certain strategies that would require non null/NA values in order for them to be able to calculate the given metrics correctly.

As can be seen, indeed there are a few Null values in our dataset.

We will drop the NA values and use the linear interpolation method so that we are able to download all the strategies correctly from the Pandas TA library without receiving any error. In addition to avoiding the error, the cleaning will also help in avoiding any misguided suggestions (although there is sufficient amount of data and the null values are relatively less compared to the total data present.)

```
df.isnull().sum()
Open        14
High        14
Low         14
Close       14
Adj Close   14
Volume      14
dtype: int64

#check for null values in dataset
df.isnull()
```

	Open	High	Low	Close	Adj Close	Volume
Date						
13-01-2014	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
14-01-2014	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
15-01-2014	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
16-01-2014	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
17-01-2014	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
...	...	...	...	...	...	...
08-01-2024	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
09-01-2024	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
10-01-2024	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
11-01-2024	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
12-01-2024	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

2469 rows × 6 columns

```
# drop null values
df.dropna()

# linear interpolation
df.interpolate(method='linear', axis=0, inplace = True)
```

Feature Engineering

The pandas TA library has been used in order to generate features. All the strategies from the given library have been imported so that we have a vast variety of features to choose from in our feature selection process.

```
# add all factors from pandas TA library
df.ta.strategy('All')
```

We have dropped some unwanted column from the set that has been downloaded as they wouldnt be relevant for our analysis. We are segregating the features and our predicition i.e. the X variables (inputs) and the Y variable (output). Next, we split the data into our training and testing sets for X and Y respectively using the train_test_split function from the scikitlearn library.

In addition to the above, we also define our labels.

Label Definition:

Label or the target variable is also known as the dependent variable. Here, the target variable is whether Nifty50 Index price will close up or down on the next bar. If the next bar closing price is greater than current bar closing price, then we will buy the Nifty50 Index, else do nothing. We assign a value of +1 for the buy signal, else 0 to target variable.

The target can be described as:

$$Y = \begin{cases} +1 & \text{if } Pt+1 > Pt \\ 0, & \text{Otherwise} \end{cases}$$

```
# copy dataframe
data = df.copy()

# define target (label)
data['predict'] = np.where(data['Adj Close'].pct_change(-1) > 0, 1, 0)

# drop unwanted columns
data.drop(['HILOl_13_21', 'HILOs_13_21', 'PSARl_0.02_0.2', 'PSARs_0.02_0.2', 'PSARaf_0.02_0.2', 'QQEl_14_5_4.236', 'QQEs_14_5_4.236', 'SUPERTl_7_3.0', 'SUPERTs_7_3.0'], axis=1, inplace=True)
data = data[200:]

# backfill columns to address missing values
data = data.bfill(axis=1)

# check last 5 rows
data.tail()
```

	Open	High	Low	Close	Adj Close	Volume	ABER_ZG_5_15	ABER_SG_5_15	ABER_XG_5_15	ABER_ATR_5_15	...	VTXP_14	VTXM_14	VWAP_D	VWMA_10	WCP	WILLR_14	WMA_10	ZL_EMA_10	ZS_30	predict
Date																					
2024-01-01	21744.50000	21762.950195	21616.27539	21698.600585	21698.600585	267300.0	21639.255078	21824.309420	21454.200736	185.054342	...	1.136044	0.770625	21692.608723	21493.632135	21694.106689	-9.966443	21574.037393	21768.783939	1.288428	1.0

	Open	High	Low	Close	Adj Close	Volume	ABER_ZG_5_15	ABER_SG_5_15	ABER_XG_5_15	ABER_ATR_5_15	...	VTXP_14	VTXM_14	VWAP_D	VWMA_10	WCP	WILLR_14	WMA_10	ZL_EMA_10	ZS_30	predict
Date																					
2024-01-02	21751.34961	21755.599610	21555.65039	21665.800780	21665.800780	263700.0	21687.861849	21873.909182	21501.814515	186.047333	...	1.133372	0.779208	21659.016927	21516.156218	21660.712890	-13.144875	21605.434783	21752.068961	1.170128	1.0
2024-01-03	21661.09961	21677.000000	21500.34961	21517.349610	21517.349610	311900.0	21679.088411	21864.509282	21493.667541	185.420871	...	1.147621	0.775833	21564.899740	21521.698934	21553.012208	-34.450997	21605.347407	21661.874604	0.901608	0.0
2024-01-04	21605.80078	21685.650390	21564.55078	21658.599610	21658.599610	339200.0	21655.798567	21840.078098	21471.519036	184.279531	...	1.105195	0.802872	21636.266927	21584.675470	21641.850097	-17.322486	21629.773668	21648.042645	1.055494	0.0
2024-01-05	21705.75000	21749.599610	21629.19922	21710.800780	21710.800780	309300.0	21649.865104	21829.886026	21469.844182	180.020922	...	1.096426	0.870938	21696.533203	21629.420647	21700.100097	-10.992374	21654.446520	21661.671433	1.081911	1.0

5 rows × 216 columns

 # class frequency
c = data['predict'].value_counts()
c

 # class weight function
def cwts(dfs):
c0, c1 = np.bincount(dfs['predict'])
w0=(1/c0)*(len(df))/2
w1=(1/c1)*(len(df))/2
return {0: w0, 1: w1}

 # check class weights
class_weight = cwts(data)
class_weight

 # With the calculated weights, both classes gain equal weight
class_weight[0] * c[0], class_weight[1] * c[1]

(1234.5, 1234.5)

X = data.drop('predict', axis=1)
feature_names = X.columns
X.tail()

	Open	High	Low	Close	Adj Close	Volume	ABER_ZG_5_15	ABER_SG_5_15	ABER_XG_5_15	ABER_ATR_5_15	...	VIDYA_14	VTXP_14	VTXM_14	VWAP_D	VWMA_10	WCP	WILLR_14	WMA_10	ZL_EMA_10	ZS_30
Date																					
2024-01-01	21744.50000	21762.950195	21616.27539	21698.600585	21698.600585	267300.0	21639.255078	21824.309420	21454.200736	185.054342	...	21101.381565	1.136044	0.770625	21692.608723	21493.632135	21694.106689	-9.966443	21574.037393	21768.783939	1.288428

	Open	High	Low	Close	Adj Close	Volume	ABER_ZG_5_15	ABER_SG_5_15	ABER_XG_5_15	ABER_ATR_5_15	...	VIDYA_14	VTXP_14	VTXM_14	VWAP_D	VWMA_10	WCP	WILLR_14	WMA_10	ZL_EMA_10	ZS_30
Date																					
2024-01-02	21751.34961	21755.599610	21555.65039	21665.800780	21665.800780	263700.0	21687.861849	21873.909182	21501.814515	186.047333	...	21135.662354	1.133372	0.779208	21659.016927	21516.156218	21660.712890	-13.144875	21605.434783	21752.068961	1.170128
2024-01-03	21661.09961	21677.000000	21500.34961	21517.349610	21517.349610	311900.0	21679.088411	21864.509282	21493.667541	185.420871	...	21152.412741	1.147621	0.775833	21564.899740	21521.698934	21553.012208	-34.450997	21605.347407	21661.874604	0.901608
2024-01-04	21605.80078	21685.650390	21564.55078	21658.599610	21658.599610	339200.0	21655.798567	21840.078098	21471.519036	184.279531	...	21171.525688	1.105195	0.802872	21636.266927	21584.675470	21641.850097	-17.322486	21629.773668	21648.042645	1.055494
2024-01-05	21705.75000	21749.599610	21629.19922	21710.800780	21710.800780	309300.0	21649.865104	21829.886026	21469.844182	180.020922	...	21184.053049	1.096426	0.870938	21696.533203	21629.420647	21700.100097	-10.992374	21654.446520	21661.671433	1.081911

5 rows × 215 columns

```
y = data['predict'].values
y = y.astype(int)
y
```

```
array([0, 0, 1, ..., 0, 0, 1])
```

Feature Selection:
Bortua algorithm is designed to take the "all-relevant" approach to feature selection. It is a wrapper built around the random forest classification algorithm which is relatively quick and can run without tuning of parameters and it gives a numerical estimate of the feature importance. Importance measure of an attribute is obtained as the loss of accuracy of classification caused by the random permutation of attribute values between objects.

```
# Always keep shuffle = False for financial time series
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, shuffle=False)

# convert to array
X_train, X_test, y_train, y_test = np.array(X_train),
np.array(X_test), np.array(y_train), np.array(y_test)

# Output the train and test data size
print(f"Train and Test Size {len(X_train)}, {len(X_test)}")

# define random forest classifier
forest = RandomForestClassifier(n_jobs=-1,
                                class_weight=cwts(data),
                                random_state=42,
                                max_depth=5)

# train the model
forest.fit(X_train, y_train)
```

Train and Test Size 1811, 453

```
RandomForestClassifier(class_weight={0: 1.009403107113
6549,
                                     1: 1.185878962536023},
                       max_depth=5, n_jobs=-1, random_state=42)


# define Boruta feature selection method
feat_selector = BorutaPy(forest, n_estimators='auto',
alpha=0.15, verbose=2, random_state=0)

# takes input in array format not as dataframe
feat_selector.fit(X_train, y_train)

# check selected features
print(feat_selector.support_)

# check ranking of features
print(feat_selector.ranking_)

# call transform() on X to filter it down to selected features
X_filtered = feat_selector.transform(X_train)
```

```
Iteration:   97 / 100
Confirmed:              5
Tentative:  1
Rejected:   209
Iteration:   98 / 100
Confirmed:              5
Tentative:  1
Rejected:   209
Iteration:   99 / 100
Confirmed:              5
Tentative:  1
Rejected:   209

BorutaPy finished running.

Iteration:   100 / 100
Confirmed:              5
Tentative:  0
Rejected:   209
[False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False  True False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False  True False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False  True False False False False False False False False False
 False False False False False False  True False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False  True False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False]
[178 181 150 152 165  41 173 153 132  67 145 127 137  80  14  57  17   7
 136 204 202 106  74 133 124 103 100 198 205  53 190 187 174  48 164 170
 139  31   1  21   3  42  22  40 199 195 109  63  69  32   2  77  78 119
 130  12  14  91 117 162 146 168 144 185 166   1   5   9 139  85  38  59
   8  28  99 114 158 128 182 178 192 142 189 188 178 115 123 176 159 121
 146 142 172   4 171  57 185 116 149 154 134  49 105  16  23  25  20  52
  44 168   1  50  83  82  81  39 159 148  70 174 156  70  68  28 193  13
   1  55  97  73  84 194 183  33  18  36  64 103 118 120 139  95  76  46
 139 157  86  18  98  28  35  24 162  36   1 113 109 102  33  42 197 202
 210  53 200 208 209 207 210 180 121 112 125  46  89  60  95  72  86 202
 167 135 154  30  11 206 196  66 106  44  51  88  55 101 128  94  64  10
  79  91 185 108  60  62  93 111  90  75 162 131 151   6 126 191  26]
```

```
# zip my names, ranks, and decisions in a single iterable
feature_ranks = list(zip(feature_names,
                         feat_selector.ranking_,
                         feat_selector.support_))

# iterate through and print out the results
for feat in feature_ranks:
    print(f'Feature: {feat[0]:<30} Rank: {feat[1]:<5} Keep: {feat[2]}')
```

```
Feature: Open                  Rank: 178   Keep: False
Feature: High                  Rank: 181   Keep: False
.
.
Feature: WMA_10                Rank: 126   Keep: False
Feature: ZL_EMA_10             Rank: 191   Keep: False
Feature: ZS_30                 Rank: 26    Keep: False
```

```
selected_rf_features =
pd.DataFrame({'Feature':feature_names,
              'Ranking':feat_selector.ranking_})

# selected_rf_features#.sort_values(by='Ranking')
```

```
selected_rf_features[selected_rf_features['Ranking']==1]
```

	Feature	Ranking
38	BBP_5_2.0	1
65	DPO_20	1
110	LOGRET_1	1
126	PCTRET_1	1
154	SLOPE_1	1

It turns out that only 6 features are relevant for our analysis as per the BorutaPy selection algortihm.

```
# check the shape
X_filtered.shape

# first apply feature selector transform to make sure same features are selected
X_test_filtered = feat_selector.transform(X_test)
X_test_filtered
```

```
array([[ 6.66932756e-01, -3.72055469e+02,  2.05144228e-02,
         2.07262898e-02,  3.31899410e+02],
```

```
[ 8.74099646e-01, -1.71207179e+02,  1.51520134e-02,
  1.52673871e-02,  2.49550780e+02],
[ 7.77865108e-01, -1.69003368e+02,  2.13986272e-03,
  2.14215386e-03,  3.55488300e+01],
...,
[ 4.63646048e-02,  6.23277126e-01, -6.87544811e-03,
 -6.85186629e-03, -1.48451170e+02],
[ 5.14507084e-01, -4.87307111e-03,  6.54301788e-03,
  6.56447019e-03,  1.41250000e+02],
[ 7.18664184e-01, -4.89694117e-01,  2.40728231e-03,
  2.41018214e-03,  5.22011700e+01]])
```

Feature Scaling:

The idea of feature scaling is to make sure that features are on a similar scale. I have used the MinMaxScaler as the scaler for the features that have been selected via the BorutaPy algorithm. This step is important because our features might differ from each other (in absolute terms) quite a lot (e.g. prices and returns). Scaling increases chances to converge to the optimum (faster).

```
# perform normalization

scaler = MinMaxScaler()
scaledtrain = scaler.fit_transform(X_filtered)
scaledtest = scaler.transform(X_test_filtered)

mkdir my_log_dir
mkdir: my_log_dir: File exists

load_ext tensorboard

os.environ['TENSORBOARD_BINARY'] =
'/Users/AumkarWagle/anaconda3/envs/python_39/bin/tensor
board'

results_path = Path('results', 'lstm_time_series')
if not results_path.exists():
    results_path.mkdir(parents=True)

# sequence length
seqlen = 30

# number of features
numfeat = scaledtrain.shape[1]
```

We now use the 'TimeseriesGenerator' from the tensorflow package to generate train and test sequence data for our training and testing datasets accordingly.

```
# generate train and test sequence data
g = TimeseriesGenerator(scaledtrain, y_train, length=seqlen)
g_ = TimeseriesGenerator(scaledtest, y_test, length=seqlen)

# verify length
len(g), len(g_)

(14, 4)

#check feature set
g[0][0]

array([[[0.82574361, 0.54084471, 0.6272708 , 0.600976  , 0.61968422],
        [0.92899249, 0.538077  , 0.6332495 , 0.60711821, 0.62573238],
        [0.93018324, 0.53679831, 0.63443076, 0.60833274, 0.62695229],
        ...,
        [0.15704369, 0.41901472, 0.60231706, 0.57542806, 0.59524818],
        [0.56112363, 0.48460235, 0.69510386, 0.67114725, 0.68599468],
        [0.76969224, 0.52345331, 0.65944002, 0.63412175, 0.65149718]],

       [[0.92899249, 0.538077  , 0.6332495 , 0.60711821, 0.62573238],
        [0.93018324, 0.53679831, 0.63443076, 0.60833274, 0.62695229],
        [0.51302725, 0.51278511, 0.6097418 , 0.58301474, 0.60194709],
        ...,
        [0.56112363, 0.48460235, 0.69510386, 0.67114725, 0.68599468],
        [0.76969224, 0.52345331, 0.65944002, 0.63412175, 0.65149718],
        [0.88490342, 0.57627347, 0.67690767, 0.65221952, 0.66934238]],

       [[0.93018324, 0.53679831, 0.63443076, 0.60833274, 0.62695229],
        [0.51302725, 0.51278511, 0.6097418 , 0.58301474, 0.60194709],
        [0.83439481, 0.51885077, 0.64053404, 0.61461305, 0.63313658],
        ...,
        [0.76969224, 0.52345331, 0.65944002, 0.63412175, 0.65149718],
        [0.88490342, 0.57627347, 0.67690767, 0.65221952, 0.66934238],
        [0.6639859 , 0.55197059, 0.59256652, 0.56548391, 0.58483394]],

       ...,

       [[0.63224943, 0.50769324, 0.682835  , 0.65837671, 0.67460393],
        [0.51362911, 0.49631276, 0.61724474, 0.5906941 , 0.60964951],
        [0.60749991, 0.51209106, 0.64412296, 0.61831006, 0.63644458],
        ...,
        [0.86643905, 0.51818884, 0.6509042 , 0.62530361, 0.64306244],
        [0.91034081, 0.56766969, 0.69269031, 0.66863224, 0.6852621 ],
```

```
 [0.8152482 , 0.57555014, 0.63861815, 0.61264067, 0.63118357]],

[[0.51362911, 0.49631276, 0.61724474, 0.5906941 , 0.60964951],
 [0.60749991, 0.51209106, 0.64412296, 0.61831006, 0.63644458],
 [0.91054719, 0.5672908 , 0.68336678, 0.6589295 , 0.67612312],
 ...,
 [0.91034081, 0.56766969, 0.69269031, 0.66863224, 0.6852621 ],
 [0.8152482 , 0.57555014, 0.63861815, 0.61264067, 0.63118357],
 [0.68715222, 0.55701312, 0.61228685, 0.5856182 , 0.60452357]],

[[0.60749991, 0.51209106, 0.64412296, 0.61831006, 0.63644458],
 [0.91054719, 0.5672908 , 0.68336678, 0.6589295 , 0.67612312],
 [0.78647617, 0.56256593, 0.6190881 , 0.59258278, 0.61141237],
 ...,
 [0.8152482 , 0.57555014, 0.63861815, 0.61264067, 0.63118357],
 [0.68715222, 0.55701312, 0.61228685, 0.5856182 , 0.60452357],
 [0.7223828 , 0.56572724, 0.64325128, 0.61741185, 0.6359027 ]]])
```

```
#check target
g[0][1]
```

```
array([1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0
,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1])
```

```
#verify batch size
for i in range(len(g)):
    a, b, = g[i]
    print(a.shape, b.shape)
```

```
(128, 30, 5) (128,)
(128, 30, 5) (128,)
(128, 30, 5) (128,)
(128, 30, 5) (128,)
(128, 30, 5) (128,)
(128, 30, 5) (128,)
(128, 30, 5) (128,)
(128, 30, 5) (128,)
(128, 30, 5) (128,)
(128, 30, 5) (128,)
(128, 30, 5) (128,)
(128, 30, 5) (128,)
(128, 30, 5) (128,)
(117, 30, 5) (117,)
```

Model Building:

As was mentioned above, I will be building three LSTM models, each with a different number of layers so that we have a variety for our analyses and can understand the influence of having multiple layers as well (or having less layers for that matter!). The model is comprised of a few elements.

A brief description of the LSTM model and it's different states is mentioned below:

- Cell state - This represents the internal memory of the cell which stores both short term memory and long-term memories.
- Hidden state - This is output state information calculated w.r.t. current input, previous hidden state and current cell input which you eventually use to predict the future stock market prices. Additionally, the hidden state can decide to only retrive the short or long-term or both types of memory stored in the cell state to make the next prediction.
- Input gate - Decides how much information from current input flows to the cell state.
- Forget gate - Decides how much information from the current input and the previous cell state flows into the current cell state.
- Output gate - Decides how much information from the current cell state flows into the hidden state, so that if needed LSTM can only pick the long-term memories or short-term memories and long-term memories

Model 1:

```
# Create a sequential model
def create_model(hu=256, lookback=60, features=1):

    tf.keras.backend.clear_session()

    # instantiate the model
    model = Sequential()

    model.add(LSTM(units=hu*2, input_shape=(lookback,
features), activation = 'elu', return_sequences=True,
name='LSTM1'))
    model.add(Dropout(0.4, name='Drouput1'))

    model.add(LSTM(units=hu, activation = 'elu',
return_sequences=True, name='LSTM2'))
    model.add(Dropout(0.4, name='Drouput2'))

    model.add(LSTM(units=hu, activation = 'elu',
return_sequences=False, name='LSTM3'))

    model.add(Dense(units=1, activation='sigmoid',
name='Output'))

    # specify optimizer separately (preferred method))
    opt = Adam(lr=0.001, epsilon=1e-08, decay=0.0)

    # model compilation - 'binary_crossentropy' - 'accuracy' -
BinaryAccuracy(name='accuracy', threshold=0.5)
```

```
    model.compile(optimizer=opt,
                  loss=BinaryCrossentropy(),
                  metrics=['accuracy',
                           Precision(),
                           Recall()])

    return model
```

The architecture of the above model is to have three LSTM layers, the first two layers have a dropout layer. FInally the output layer which is the dense layer.

```
# lstm network
model = create_model(hu=10, lookback=seqlen,
features=numfeat)

# summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
LSTM1 (LSTM)	(None, 30, 20)	2080
Drouput1 (Dropout)	(None, 30, 20)	0
LSTM2 (LSTM)	(None, 30, 10)	1240
Drouput2 (Dropout)	(None, 30, 10)	0
LSTM3 (LSTM)	(None, 10)	840
Output (Dense)	(None, 1)	11

Total params: 4,171
Trainable params: 4,171
Non-trainable params: 0

```
# Specify callback functions
model_path = (results_path / 'model.h5').as_posix()
logdir = os.path.join("./tensorboard/logs",
dt.datetime.now().strftime("%Y%m%d-%H%M%S"))

my_callbacks = [
    EarlyStopping(patience=20, monitor='loss', mode='min',
verbose=1, restore_best_weights=True),
    ModelCheckpoint(filepath=model_path, verbose=1,
monitor='loss', save_best_only=True),
    TensorBoard(log_dir=logdir, histogram_freq=1)
]
```

```
2024-01-19 13:52:44.642201: I tensorflow/core/profiler/lib/
profiler_session.cc:131] Profiler session initializing.
2024-01-19 13:52:44.642728: I tensorflow/core/profiler/lib/
profiler_session.cc:146] Profiler session started.
2024-01-19 13:52:44.645184: I tensorflow/core/profiler/lib/
profiler_session.cc:164] Profiler session tear down.
```

In [167]:

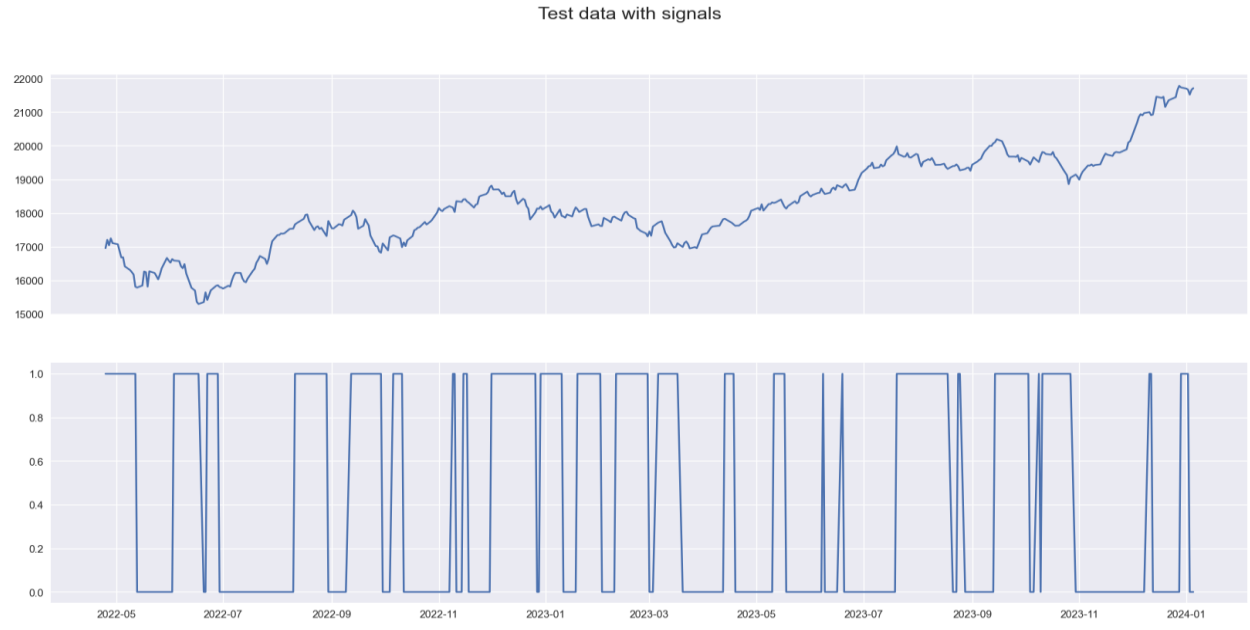
```
# Model fitting
model.fit(g,
          epochs=500,
          verbose=1,
          callbacks=my_callbacks,
          shuffle=False,
          class_weight=class_weight)
```

```
Epoch 1/500
 2/14 [===>.....................] - ETA: 5s - loss: 0.7545 - accu
racy: 0.5352 - precision: 0.5263 - recall: 0.9848
2024-01-19 13:52:57.555987: I tensorflow/core/profiler/lib/
profiler_session.cc:131] Profiler session initializing.
2024-01-19 13:52:57.556028: I tensorflow/core/profiler/lib/
profiler_session.cc:146] Profiler session started.
2024-01-19 13:52:58.558822: I tensorflow/core/profiler/lib/
profiler_session.cc:66] Profiler session collecting data.
2024-01-19 13:52:58.691115: I tensorflow/core/profiler/lib/
profiler_session.cc:164] Profiler session tear down.
2024-01-19 13:52:58.779665: I tensorflow/core/profiler/rpc/
client/save_profile.cc:136] Creating directory: ./tensorboard/
logs/20240119-135244/train/plugins/profile/2024_01_19_13
_52_58

2024-01-19 13:52:58.826066: I tensorflow/core/profiler/rpc/
client/save_profile.cc:142] Dumped gzipped tool data for tra
ce.json.gz to ./tensorboard/logs/20240119-135244/train/plug
ins/profile/2024_01_19_13_52_58/Aumkars-MacBook-Air.l
ocal.trace.json.gz
2024-01-19 13:52:58.908375: I tensorflow/core/profiler/rpc/
client/save_profile.cc:136] Creating directory: ./tensorboard/
logs/20240119-135244/train/plugins/profile/2024_01_19_13
_52_58

14/14 [==============================] - 10s 208
ms/step - loss: 0.7580 - accuracy: 0.4902 - precision: 0.4657
- recall: 0.7293

Epoch 00001: loss improved from inf to 0.75801, saving mo
del to results/lstm_time_series/model.h5
Epoch 2/500
14/14 [==============================] - 1s 73ms
/step - loss: 0.7589 - accuracy: 0.4891 - precision: 0.4074 - r
ecall: 0.2415

Epoch 00002: loss did not improve from 0.75801
.
.
.

Epoch 00191: loss did not improve from 0.65780
Epoch 192/500
14/14 [==============================] - 1s 68ms
/step - loss: 0.6657 - accuracy: 0.6749 - precision: 0.6471 - r
ecall: 0.6463

Epoch 00192: loss did not improve from 0.65780
Epoch 193/500
14/14 [==============================] - 1s 66ms
/step - loss: 0.6590 - accuracy: 0.6682 - precision: 0.6375 - r
ecall: 0.6476
Restoring model weights from the end of the best epoch.

Epoch 00193: loss did not improve from 0.65780
Epoch 00193: early stopping

<keras.callbacks.History at 0x7fa6a7fa7e80>
```

```
# predictions
ypred = np.where(model.predict(g_, verbose=False) > 0.5, 1, 0)
ypred[-10:]
```

```
array([[0],
       [0],
       [0],
       [0],
       [1],
       [1],
       [1],
       [0],
       [0],
       [0]])
```

```
# plot predictions
df1 = data['Adj Close'][-(len(scaledtest)-seqlen):]
fig, axs = plt.subplots(2, sharex=True, figsize=(20,10))
fig.suptitle('Test data with signals', size=18)
axs[0].plot(df1.index, df1)
axs[1].plot(df1.index, ypred);
```



We will now evaluate the model that we have created.

```
# load model - os.path.abspath(model_path)
model = load_model(model_path)

# summarize model
model.summary()

# evaluate the model
score = model.evaluate(g_, verbose=0)
print(f'{model.metrics_names[1]}, {score[1]*100:.4}%')
```

Model: "sequential"

Layer (type)	Output Shape	Param #
LSTM1 (LSTM)	(None, 30, 20)	2080
Drouput1 (Dropout)	(None, 30, 20)	0
LSTM2 (LSTM)	(None, 30, 10)	1240
Drouput2 (Dropout)	(None, 30, 10)	0
LSTM3 (LSTM)	(None, 10)	840
Output (Dense)	(None, 1)	11

Total params: 4,171
Trainable params: 4,171
Non-trainable params: 0

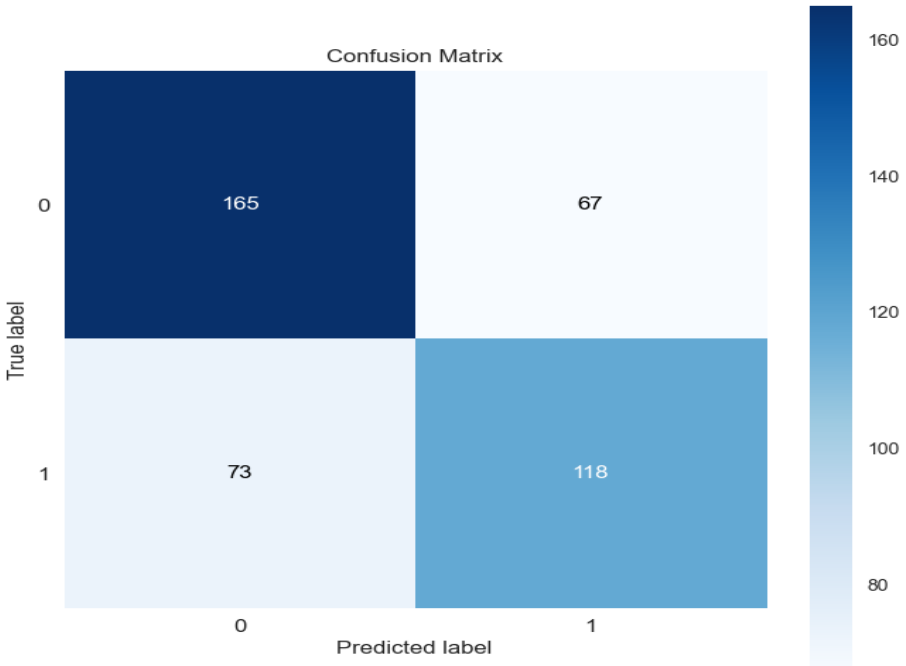
accuracy, 66.9%

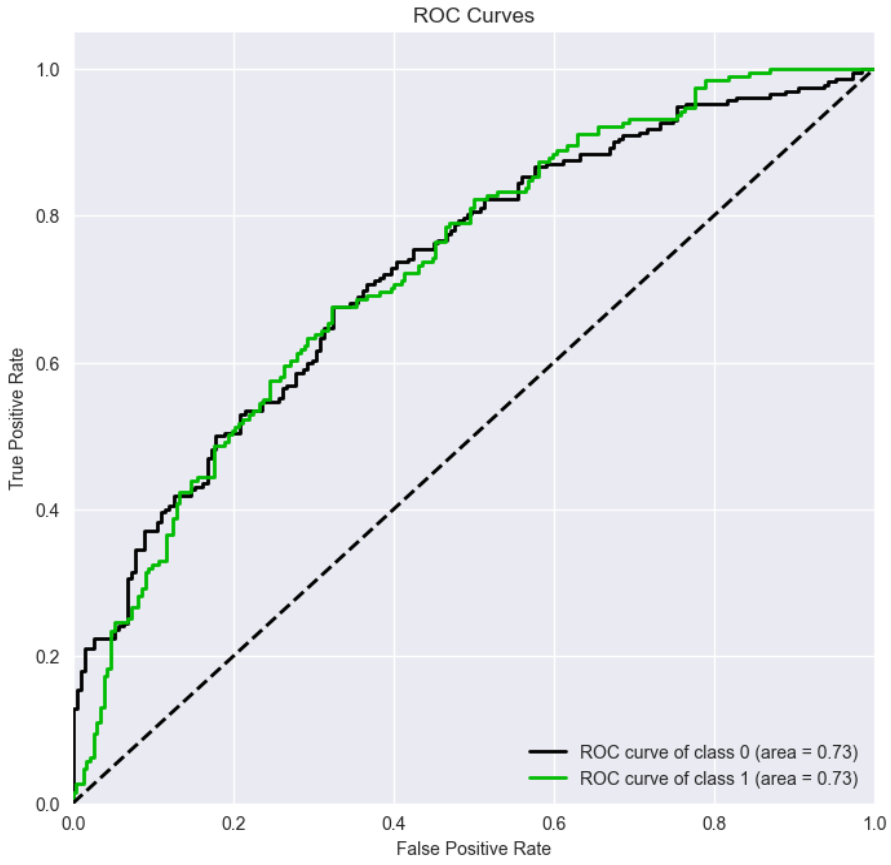
Now I have created a function that will allow us to visualize and view how well our model has performed using the Confusion Matrix, ROC Curve and Classification Report. This function will also be used later on to evaluate and visualize the other two models that are created in this project.

```
def get_model_results(model, g_, y_test, threshold=0.5):
    """
    Generates the Confusion Matrix, ROC Curve, and Classification Report for a keras model
    """

    Y_test_n = y_test[-1*model.predict(g_).shape[0]:]
    Y_pred = np.where(model.predict(g_, verbose=False) > threshold, 1, 0)
    plot_confusion_matrix(Y_test_n, Y_pred, normalize = False, figsize=(8,8), text_fontsize='large')

    prob_df = pd.DataFrame(model.predict(g_), columns=['Prob_1'])
    prob_df['Prob_0'] = 1 - prob_df['Prob_1']
    prob_array = prob_df.loc[:,['Prob_0', 'Prob_1']].to_numpy()

    plot_roc(Y_test_n, prob_array, plot_micro = False, plot_macro = False, figsize=(8,8))

    print("Classification Report: \n")
```

```
    print(classification_report(Y_test_n, Y_pred,
zero_division=0))

    return
get_model_results(model, g_,y_test, threshold=0.5)
Classification Report:
```

	precision	recall	f1-score	support
0	0.69	0.71	0.70	232
1	0.64	0.62	0.63	191
accuracy			0.67	423
macro avg	0.67	0.66	0.66	423
weighted avg	0.67	0.67	0.67	423





Model 2:
The artchitecture of this model is rather simple i.e. we just have one LSTM layer which has a dropout layer and the output layer which is the Dense layer. The reason for this is to see how the algorithm performs relative to a model that is not as shallow as this one.

```
# Create a sequential model
def create_model_2(hu=256, lookback=60, features=1):

    tf.keras.backend.clear_session()

    # instantiate the model
    model_2 = Sequential()

    model_2.add(LSTM(units=hu*2, input_shape=(lookback, features), activation = 'elu', return_sequences=False, name='LSTM1'))
    model_2.add(Dropout(0.4, name='Drouput1'))

    model_2.add(Dense(units=1, activation='sigmoid', name='Output'))

    # specify optimizer separately (preferred method))
    opt = Adam(lr=0.001, epsilon=1e-08, decay=0.0)

    # model compilation - 'binary_crossentropy' - 'accuracy' - BinaryAccuracy(name='accuracy', threshold=0.5)
    model_2.compile(optimizer=opt,
                    loss=BinaryCrossentropy(),
                    metrics=['accuracy',
                             Precision(),
                             Recall()])

    return model_2

# lstm network
model_2 = create_model_2(hu=10, lookback=seqlen, features=numfeat)

# summary
model_2.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
LSTM1 (LSTM)	(None, 20)	2080
Drouput1 (Dropout)	(None, 20)	0
Output (Dense)	(None, 1)	21

Total params: 2,101
Trainable params: 2,101
Non-trainable params: 0

```
# Specify callback functions
model_path_2 = (results_path / 'model_2.h5').as_posix()
logdir = os.path.join("./tensorboard/logs", dt.datetime.now().strftime("%Y%m%d-%H%M%S"))
my_callbacks = [
    EarlyStopping(patience=20, monitor='loss', mode='min', verbose=1, restore_best_weights=True),
    ModelCheckpoint(filepath=model_path_2, verbose=1, monitor='loss', save_best_only=True),
    TensorBoard(log_dir=logdir, histogram_freq=1)
]
```

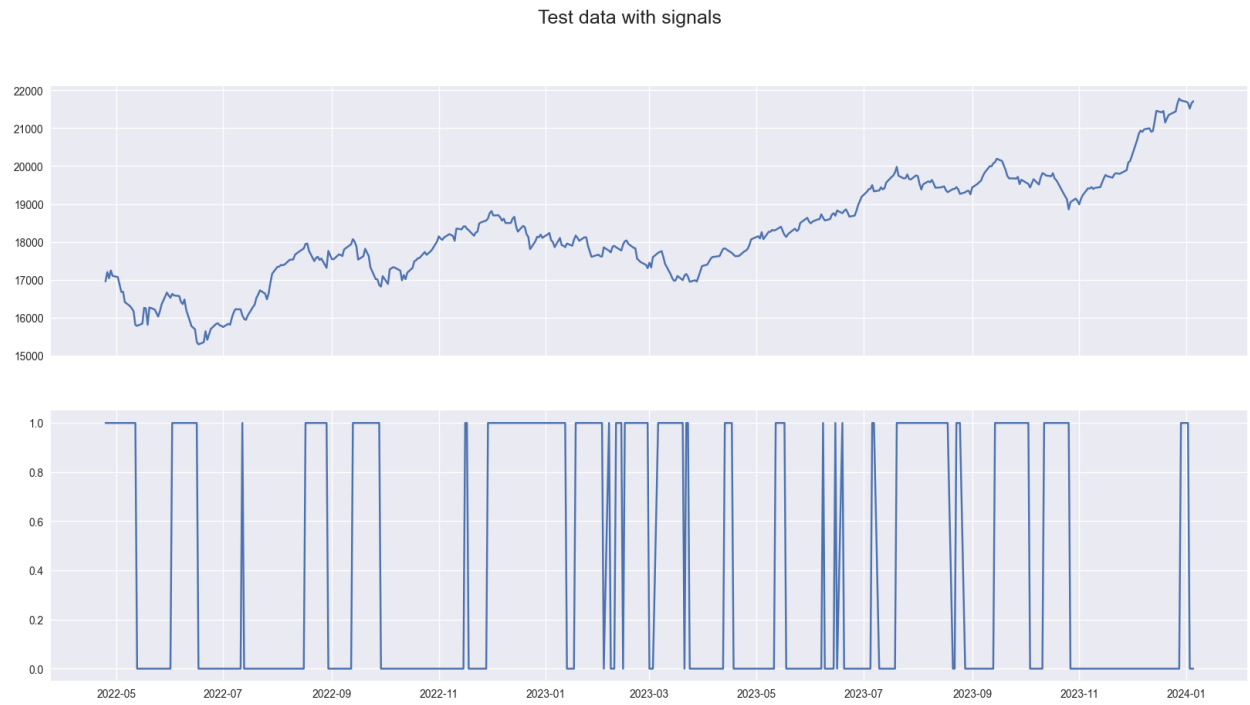
```
2024-01-19 14:00:32.755416: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2024-01-19 14:00:32.755466: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
2024-01-19 14:00:32.767298: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.
```

```
# Model fitting
model_2.fit(g,
          epochs=500,
          verbose=1,
          callbacks=my_callbacks,
          shuffle=False,
          class_weight=class_weight)
```

```
Epoch 1/500
 2/14 [===>.....] - ETA: 2s - loss: 0.8227 - accuracy: 0.5039 - precision: 0.6000 - recall: 0.1136
2024-01-19 14:00:39.428858: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2024-01-19 14:00:39.428896: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
 3/14 [=====>.....] - ETA: 4s - loss: 0.8048 - accuracy: 0.5000 - precision: 0.4681 - recall: 0.1164
2024-01-19 14:00:39.939014: I tensorflow/core/profiler/lib/profiler_session.cc:66] Profiler session collecting data.
2024-01-19 14:00:39.991100: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.
2024-01-19 14:00:40.029059: I tensorflow/core/profiler/rpc/client/save_profile.cc:136] Creating directory: ./tensorboard/logs/20240119-140032/train/plugins/profile/2024_01_19_14_00_39

2024-01-19 14:00:40.048141: I tensorflow/core/profiler/rpc/client/save_profile.cc:142] Dumped gzipped tool data for trace.json.gz to ./tensorboard/logs/20240119-140032/train/plugins/profile/2024_01_19_14_00_39/Aumkars-MacBook-Air.local.trace.json.gz
2024-01-19 14:00:40.083437: I tensorflow/core/profiler/rpc/client/save_profile.cc:136] Creating directory: ./tensorboard/logs/20240119-140032/train/plugins/profile/2024_01_19_14_00_39

2024-01-19 14:00:40.083818: I tensorflow/core/profiler/rpc/client/save_profile.cc:142] Dumped gzipped tool data for memory_profile.json.gz to ./tensorboard/logs/20240119-140032/train/plugins/profile/2024_01_19_14_00_39/Aumkars-MacBook-Air.local.memory_profile.json.gz
2024-01-19 14:00:40.090444: I tensorflow/core/profiler/rpc/client/capture_profile.cc:251] Creating directory: ./tensorboard/logs/20240119-140032/train/plugins/profile/2024_01_19_14_00_39
Dumped tool data for xplane.pb to ./tensorboard/logs/20240119-140032/train/plugins/profile/2024_01_19_14_00_39/Aumkars-MacBook-Air.local.xplane.pb
```

```
Dumped tool data for overview_page.pb to ./tensorboard/log
s/20240119-140032/train/plugins/profile/2024_01_19_14_0
0_39/Aumkars-MacBook-Air.local.overview_page.pb
Dumped tool data for input_pipeline.pb to ./tensorboard/logs
/20240119-140032/train/plugins/profile/2024_01_19_14_00
_39/Aumkars-MacBook-Air.local.input_pipeline.pb
Dumped tool data for tensorflow_stats.pb to ./tensorboard/lo
gs/20240119-140032/train/plugins/profile/2024_01_19_14_
00_39/Aumkars-MacBook-Air.local.tensorflow_stats.pb
Dumped tool data for kernel_stats.pb to ./tensorboard/logs/2
0240119-140032/train/plugins/profile/2024_01_19_14_00_3
9/Aumkars-MacBook-Air.local.kernel_stats.pb

14/14 [==============================] - 5s 84ms
/step - loss: 0.7770 - accuracy: 0.5070 - precision: 0.4517 - r
ecall: 0.3305

Epoch 00001: loss improved from inf to 0.77700, saving mo
del to results/lstm_time_series/model_2.h5
Epoch 2/500
14/14 [==============================] - 0s 21ms
/step - loss: 0.7682 - accuracy: 0.4891 - precision: 0.4534 - r
ecall: 0.5341

Epoch 00002: loss improved from 0.77700 to 0.76817, savin
g model to results/lstm_time_series/model_2.h5
.
.
.
Epoch 00236: loss did not improve from 0.64759
Epoch 237/500
14/14 [==============================] - 0s 26ms
/step - loss: 0.6583 - accuracy: 0.6721 - precision: 0.6344 - r
ecall: 0.6793

Epoch 00237: loss did not improve from 0.64759
Epoch 238/500
14/14 [==============================] - 0s 23ms
/step - loss: 0.6607 - accuracy: 0.6800 - precision: 0.6420 - r
ecall: 0.6890
Restoring model weights from the end of the best epoch.

Epoch 00238: loss did not improve from 0.64759
Epoch 00238: early stopping

<keras.callbacks.History at 0x7fa6ac830c70>
```

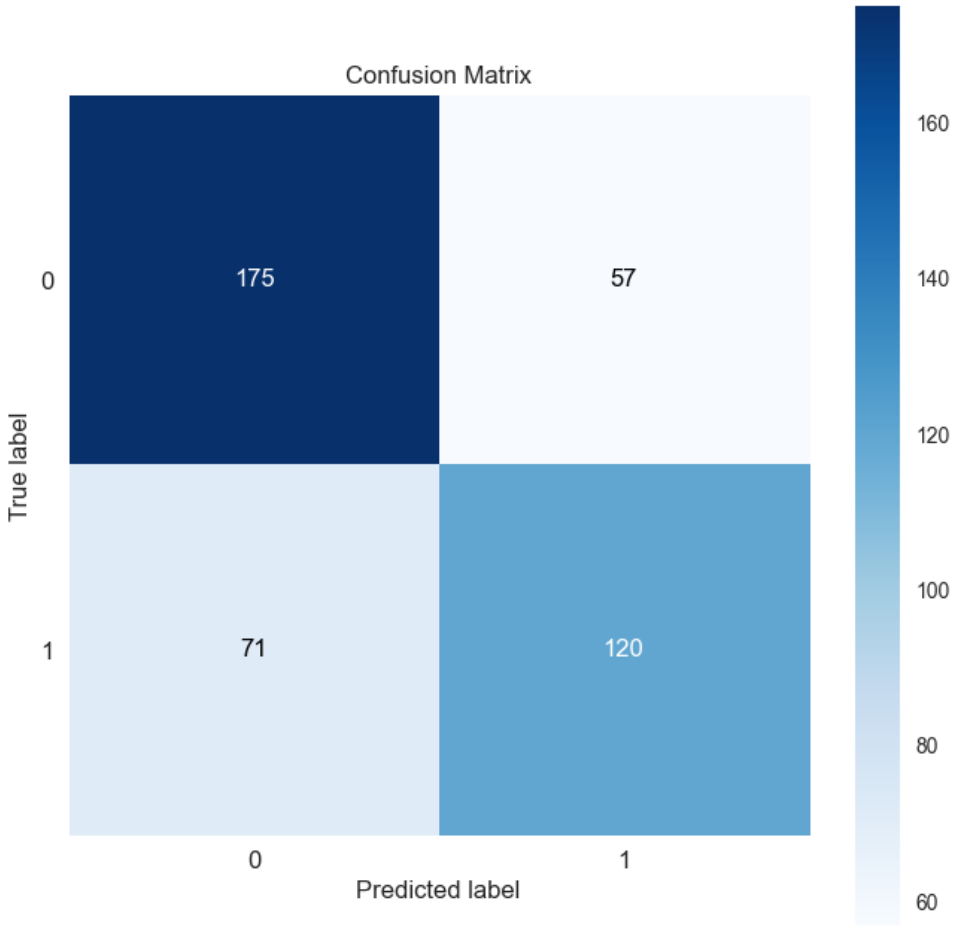
```
#predictions

ypred_2 = np.where(model_2.predict(g_, verbose=False) >
0.5, 1, 0)
ypred_2[-10:]

array([[0],
       [0],
       [0],
       [0],
       [1],
       [1],
       [1],
       [0],
       [0],
       [0]])

# plot predictions
df2 = data['Adj Close'][-(len(scaledtest)-seqlen):]
fig, axs = plt.subplots(2, sharex=True, figsize=(20,10))
fig.suptitle('Test data with signals', size=18)
axs[0].plot(df2.index, df2)
axs[1].plot(df2.index, ypred_2);
```

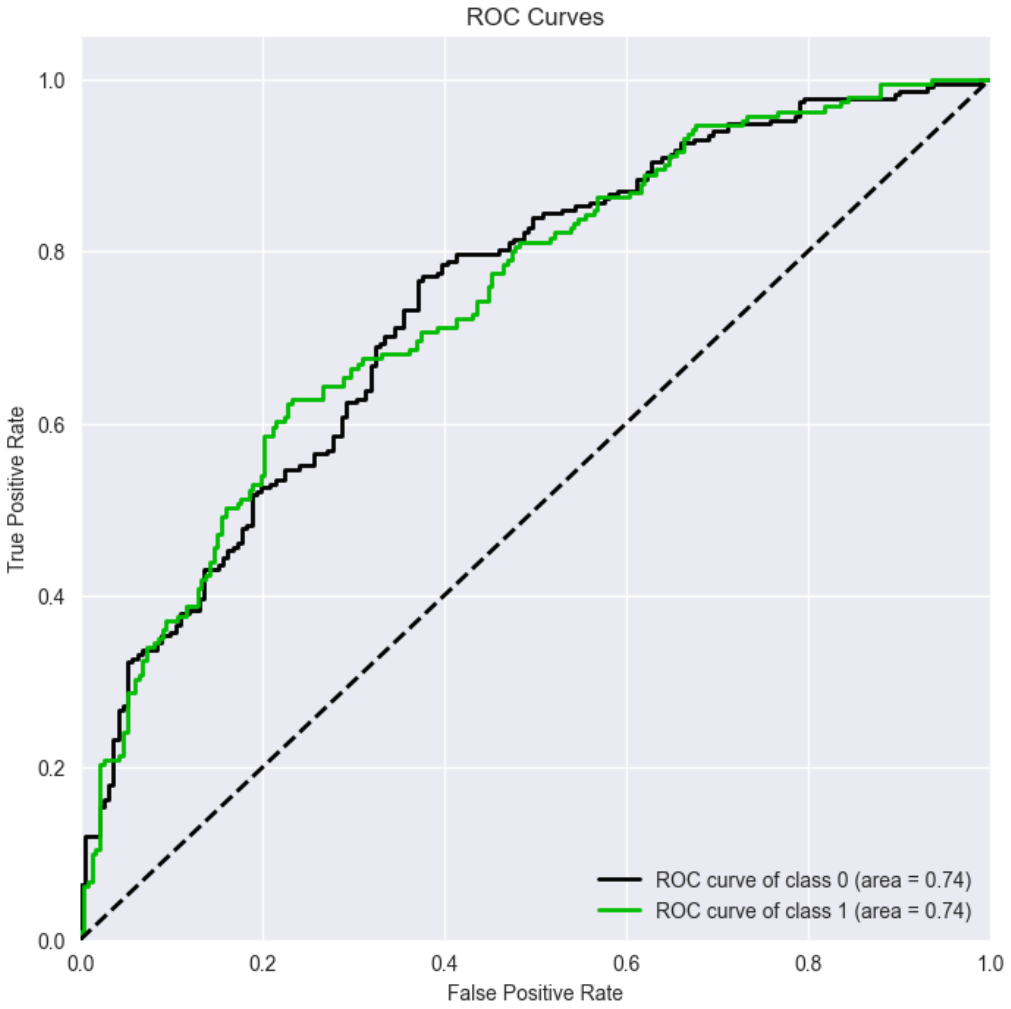



```
# load model - os.path.abspath(model_path)
model_2 = load_model(model_path_2)

# summarize model
model_2.summary()

# evaluate the model
score = model_2.evaluate(g_, verbose=0)
```

```
print(f'{model.metrics_names[1]}, {score[1]*100:.4}%')
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
LSTM1 (LSTM)                 (None, 20)                2080
_________________________________________________________________
Drouput1 (Dropout)           (None, 20)                0
_________________________________________________________________
Output (Dense)               (None, 1)                 21
=================================================================
Total params: 2,101
Trainable params: 2,101
Non-trainable params: 0
_________________________________________________________________
accuracy, 69.74%

get_model_results(model_2, g_, y_test, threshold=0.5)
Classification Report:
```

	precision	recall	f1-score	support
0	0.71	0.75	0.73	232
1	0.68	0.63	0.65	191
accuracy			0.70	423
macro avg	0.69	0.69	0.69	423
weighted avg	0.70	0.70	0.70	423





From the above metrics, we can see that this model has performed slightly better than the first model (depending on which metric we choose). However, for the most part we see that the models have performed relatively similarly, despite the difference in the number of layers.

Model 3:
The architecture of the third and final model is to have a relatively larger number of layers so that it can be compared with the earlier two models where there is a much shallower model and one with a relatively sound number of layers.
In this model, we have six LSTM layers, each of which has a dropout layer. And then we have the output layer, which is the Dense layer.

```
# Create a sequential model
def create_model_3(hu=256, lookback=60, features=1):

    tf.keras.backend.clear_session()

    # instantiate the model
    model_3 = Sequential()

    model_3.add(LSTM(units=hu*2, input_shape=(lookback, features), activation = 'elu', return_sequences=True, name='LSTM1'))
    model_3.add(Dropout(0.4, name='Drouput1'))

    model_3.add(LSTM(units=hu, activation = 'elu', return_sequences=True, name='LSTM2'))
    model_3.add(Dropout(0.4, name='Drouput2'))

    model_3.add(LSTM(units=hu, activation = 'elu', return_sequences=True, name='LSTM3'))
    model_3.add(Dropout(0.4, name='Drouput3'))

    model_3.add(LSTM(units=hu, activation = 'elu', return_sequences=True, name='LSTM4'))
    model_3.add(Dropout(0.4, name='Drouput4'))

    model_3.add(LSTM(units=hu, activation = 'elu', return_sequences=True, name='LSTM5'))
    model_3.add(Dropout(0.4, name='Drouput5'))

    model_3.add(LSTM(units=hu, activation = 'elu', return_sequences=False, name='LSTM6'))
    model_3.add(Dropout(0.4, name='Drouput6'))

    model_3.add(Dense(units=1, activation='sigmoid', name='Output'))

    # specify optimizer separately (preferred method))
    opt = Adam(lr=0.001, epsilon=1e-08, decay=0.0)

    # model compilation - 'binary_crossentropy' - 'accuracy' - BinaryAccuracy(name='accuracy', threshold=0.5)
    model_3.compile(optimizer=opt,
                    loss=BinaryCrossentropy(),
                    metrics=['accuracy',
                             Precision(),
```

```
                Recall()])

    return model_3

# lstm network
model_3 = create_model_3(hu=10, lookback=seqlen, features=numfeat)

# summary
model_3.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
LSTM1 (LSTM)	(None, 30, 20)	2080
Drouput1 (Dropout)	(None, 30, 20)	0
LSTM2 (LSTM)	(None, 30, 10)	1240
Drouput2 (Dropout)	(None, 30, 10)	0
LSTM3 (LSTM)	(None, 30, 10)	840
Drouput3 (Dropout)	(None, 30, 10)	0
LSTM4 (LSTM)	(None, 30, 10)	840
Drouput4 (Dropout)	(None, 30, 10)	0
LSTM5 (LSTM)	(None, 30, 10)	840
Drouput5 (Dropout)	(None, 30, 10)	0
LSTM6 (LSTM)	(None, 10)	840
Drouput6 (Dropout)	(None, 10)	0
Output (Dense)	(None, 1)	11

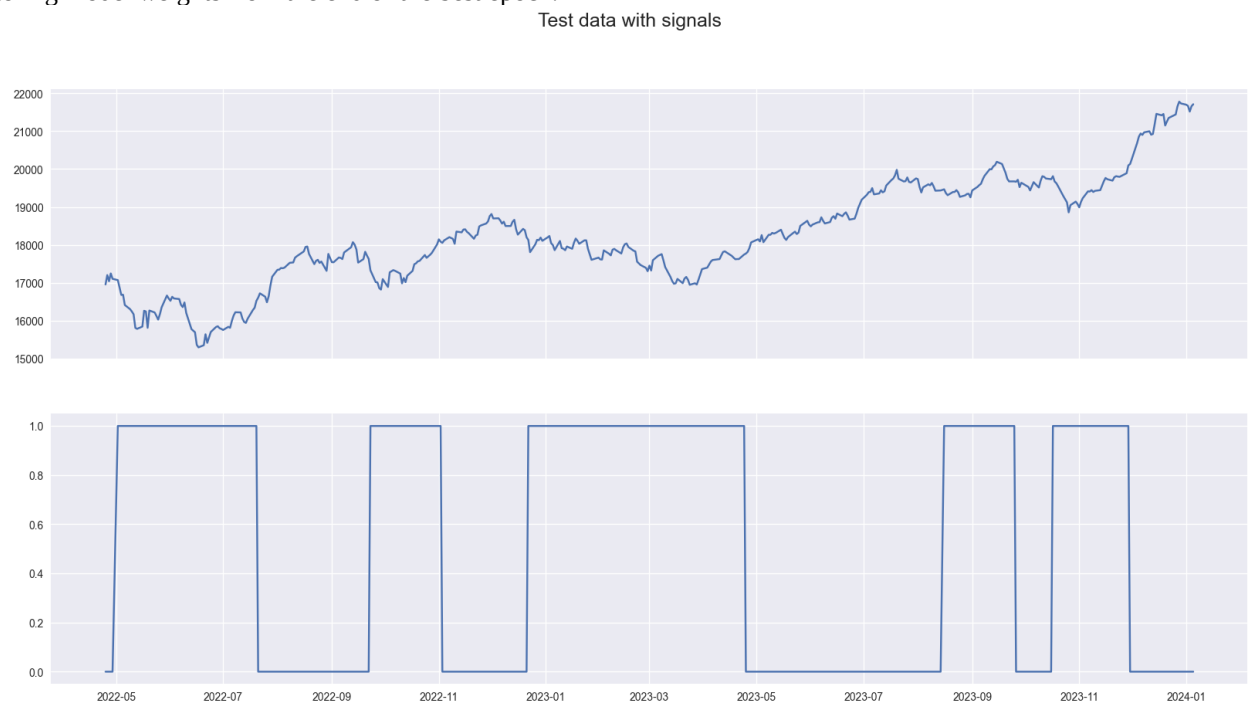
Total params: 6,691
Trainable params: 6,691
Non-trainable params: 0

```
# Specify callback functions
model_path_3 = (results_path / 'model_3.h5').as_posix()
logdir = os.path.join("./tensorboard/logs", dt.datetime.now().strftime("%Y%m%d-%H%M%S"))

my_callbacks = [
    EarlyStopping(patience=20, monitor='loss', mode='min', verbose=1, restore_best_weights=True),
    ModelCheckpoint(filepath=model_path_3, verbose=1, monitor='loss', save_best_only=True),
    TensorBoard(log_dir=logdir, histogram_freq=1)
]
2024-01-19 14:04:34.198100: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2024-01-19 14:04:34.198470: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
2024-01-19 14:04:34.199781: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.

# Model fitting
model_3.fit(g,
        epochs=500,
        verbose=1,
        callbacks=my_callbacks,
        shuffle=False,
        class_weight=class_weight)
Epoch 1/500
 2/14 [===>.....] - ETA: 11s - loss: 0.7612 - accuracy: 0.5820 - precision: 0.5969 - recall: 0.5833
2024-01-19 14:05:15.279006: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2024-01-19 14:05:15.279086: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
2024-01-19 14:05:16.279892: I tensorflow/core/profiler/lib/profiler_session.cc:66] Profiler session collecting data.
2024-01-19 14:05:16.444835: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.
2024-01-19 14:05:16.666809: I tensorflow/core/profiler/rpc/client/save_profile.cc:136] Creating directory: ./tensorboard/logs/20240119-140434/train/plugins/profile/2024_01_19_14_05_16

2024-01-19 14:05:16.744209: I tensorflow/core/profiler/rpc/client/save_profile.cc:142] Dumped gzipped tool data for trace.json.gz to ./tensorboard/logs/20240119-140434/train/plugins/profile/2024_01_19_14_05_16/Aumkars-MacBook-Air.local.trace.json.gz
 3/14 [=====>.....] - ETA: 13s - loss: 0.7593 - accuracy: 0.5521 - precision: 0.5368 - recall: 0.6561
2024-01-19 14:05:16.878719: I tensorflow/core/profiler/rpc/client/save_profile.cc:136] Creating directory: ./tensorboard/logs/20240119-140434/train/plugins/profile/2024_01_19_14_05_16

2024-01-19 14:05:16.879609: I tensorflow/core/profiler/rpc/client/save_profile.cc:142] Dumped gzipped tool data for memory_profile.json.gz to ./tensorboard/logs/20240119-140434/train/plugins/profile/2024_01_19_14_05_16/Aumkars-MacBook-Air.local.memory_profile.json.gz
14/14 [==============================] - 41s 320 ms/step - loss: 0.7561 - accuracy: 0.4896 - precision: 0.4654 - recall: 0.7293

Epoch 00001: loss improved from inf to 0.75612, saving model to results/lstm_time_series/model_3.h5
Epoch 2/500
 3/14 [=====>.....] - ETA: 1s - loss: 0.7580 - accuracy: 0.5234 - precision: 0.5124 - recall: 0.6561
2024-01-19 14:05:16.893180: I tensorflow/core/profiler/rpc/client/capture_profile.cc:251] Creating directory: ./tensorboard/logs/20240119-140434/train/plugins/profile/2024_01_19_14_05_16
Dumped tool data for xplane.pb to ./tensorboard/logs/20240119-140434/train/plugins/profile/2024_01_19_14_05_16/Aumkars-MacBook-Air.local.xplane.pb
Dumped tool data for overview_page.pb to ./tensorboard/logs/20240119-140434/train/plugins/profile/2024_01_19_14_05_16/Aumkars-MacBook-Air.local.overview_page.pb
```

```
Dumped tool data for input_pipeline.pb to ./tensorboard/logs
/20240119-140434/train/plugins/profile/2024_01_19_14_05
_16/Aumkars-MacBook-Air.local.input_pipeline.pb
Dumped tool data for tensorflow_stats.pb to ./tensorboard/lo
gs/20240119-140434/train/plugins/profile/2024_01_19_14_
05_16/Aumkars-MacBook-Air.local.tensorflow_stats.pb
Dumped tool data for kernel_stats.pb to ./tensorboard/logs/20
240119-140434/train/plugins/profile/2024_01_19_14_05_16
/Aumkars-MacBook-Air.local.kernel_stats.pb

14/14 [==============================] - 1s 103m
s/step - loss: 0.7563 - accuracy: 0.5014 - precision: 0.4687 - r
ecall: 0.6207

Epoch 00002: loss did not improve from 0.75612
Epoch 3/500
14/14 [==============================] - 2s 107m
s/step - loss: 0.7564 - accuracy: 0.4750 - precision: 0.4548 - r
ecall: 0.7049

Epoch 00003: loss did not improve from 0.75612

Epoch 00042: loss did not improve from 0.75533
Epoch 43/500
14/14 [==============================] - 1s 100m
s/step - loss: 0.7556 - accuracy: 0.4879 - precision: 0.4655 - r
ecall: 0.7573

Epoch 00043: loss did not improve from 0.75533
Epoch 44/500
14/14 [==============================] - 2s 122m
s/step - loss: 0.7556 - accuracy: 0.5065 - precision: 0.4715 - r
ecall: 0.5951
Restoring model weights from the end of the best epoch.
Epoch 00044: loss did not improve from 0.75533
Epoch 00044: early stopping

<keras.callbacks.History at 0x7fa6ad9560a0>
```

Comparing the graphs of the model structure (Graphs section of tensorboard), we can see that this model has a more complex structure than the other two but does not yield better results as can be seen in the subsequent lines of code.

```
# predictions

ypred_3 = np.where(model_3.predict(g_, verbose=False) >
0.5, 1, 0)
ypred_3[-10:]

array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0]])

# plot predictions
df3 = data['Adj Close'][-(len(scaledtest)-seqlen):]
fig, axs = plt.subplots(2, sharex=True, figsize=(20,10))
fig.suptitle('Test data with signals', size=18)
axs[0].plot(df3.index, df3)
axs[1].plot(df3.index, ypred_3);
```




```
# load model - os.path.abspath(model_path)
model_3 = load_model(model_path_3)

# summarize model
model_3.summary()
```

```
# evaluate the model
score = model_3.evaluate(g_, verbose=0)
print(f'{model.metrics_names[1]}, {score[1]*100:.4}%')
Model: "sequential"
```

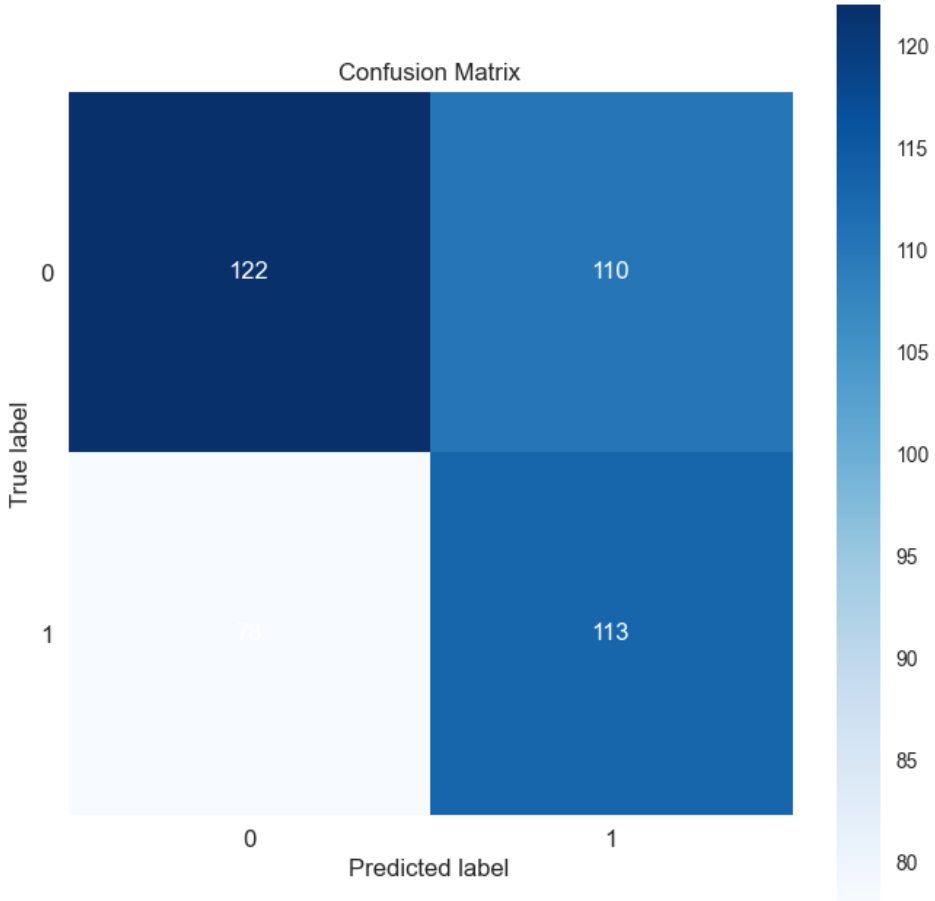
Layer (type)	Output Shape	Param #
LSTM1 (LSTM)	(None, 30, 20)	2080
Drouput1 (Dropout)	(None, 30, 20)	0
LSTM2 (LSTM)	(None, 30, 10)	1240
Drouput2 (Dropout)	(None, 30, 10)	0
LSTM3 (LSTM)	(None, 30, 10)	840
Drouput3 (Dropout)	(None, 30, 10)	0
LSTM4 (LSTM)	(None, 30, 10)	840
Drouput4 (Dropout)	(None, 30, 10)	0
LSTM5 (LSTM)	(None, 30, 10)	840
Drouput5 (Dropout)	(None, 30, 10)	0
LSTM6 (LSTM)	(None, 10)	840
Drouput6 (Dropout)	(None, 10)	0
Output (Dense)	(None, 1)	11

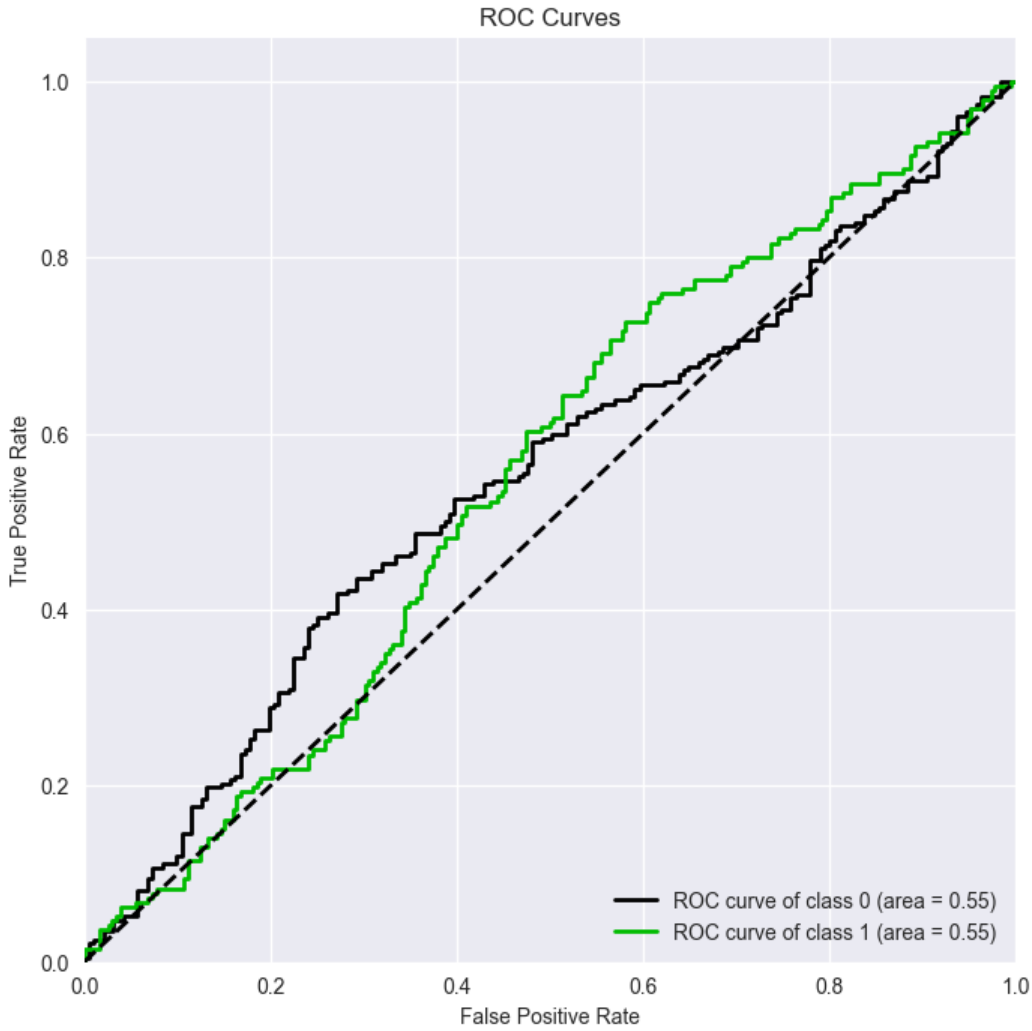
Total params: 6,691
Trainable params: 6,691
Non-trainable params: 0

accuracy, 55.56%

get_model_results(model_3, g_, y_test, threshold=0.5)
Classification Report:

	precision	recall	f1-score	support
0	0.61	0.53	0.56	232
1	0.51	0.59	0.55	191
accuracy			0.56	423
macro avg	0.56	0.56	0.56	423
weighted avg	0.56	0.56	0.56	423





As seen above, the results of this model, compared to model one and two, are underwhelming. In a relative sense, this model has underperformed both the earlier models. With an accuracy score of 55.56 compared to the mid to high 60s of the previous two models, we can say that this model is not comparabale with the other two and hence we will not be tuning it to see if it can improve further.

As an aside, out of curiosity, I did indeed test if the model performance improved after hyperparameter tuning and I noticed that the accuracy had fallen a little lower post the tuning.

Hyperparameter Tuning:
We will now jump into the hyperparameter tuning of our two best models. We use the Hyperband function of the Keras tuner to check if the parameters that we have chosen can in any way be optimized so that we can see better results for our models. This indeed does turn out to be the case as we will see in the following evaluations.

The parameters we are tuning for are the number of units in each layer/s, the dropout rate, the learning rate and the activation function.

Once we have our optimal set of parameters as will be outputted using the Hyperband tuner, we will build a subsequent model with the best parameters. Once this is done for both the models, we will then compare the validation results to see which is the best model.

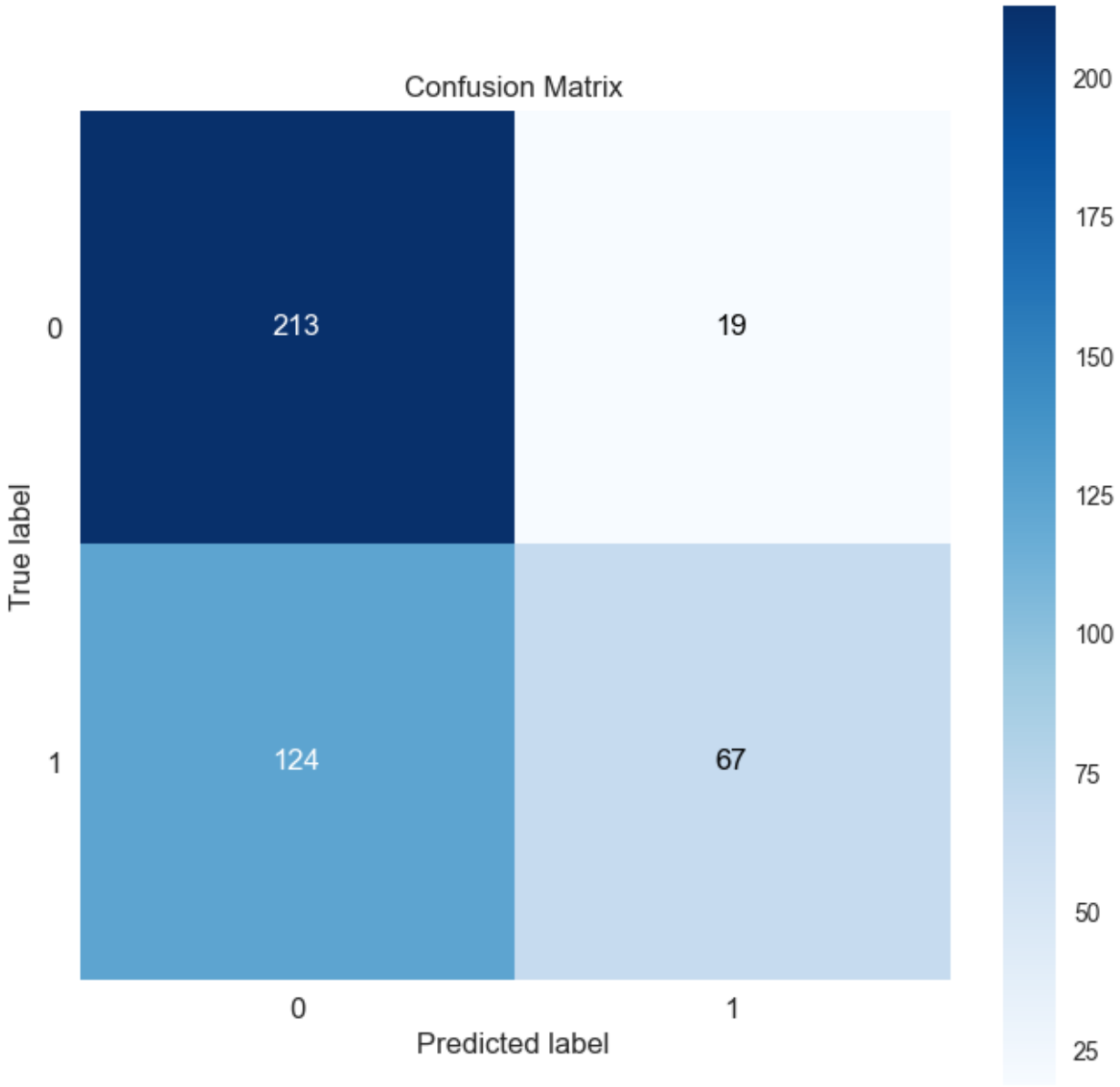
```
def build_model(hp):

    tf.keras.backend.clear_session()

    # instantiate the model
    model = Sequential()

    # Tune the number of units in the layers
    hp_units1 = hp.Int('units1', min_value=4, max_value=32, step=4)
    hp_units2 = hp.Int('units2', min_value=4, max_value=32, step=4)
    hp_units3 = hp.Int('units3', min_value=4, max_value=32, step=4)

    # Tune the dropout rate
    hp_dropout1 = hp.Float('Dropout_rate', min_value=0, max_value=0.5, step=0.1)
    hp_dropout2 = hp.Float('Dropout_rate', min_value=0, max_value=0.5, step=0.1)

    # Tune the learning rate for the optimizer
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])
```

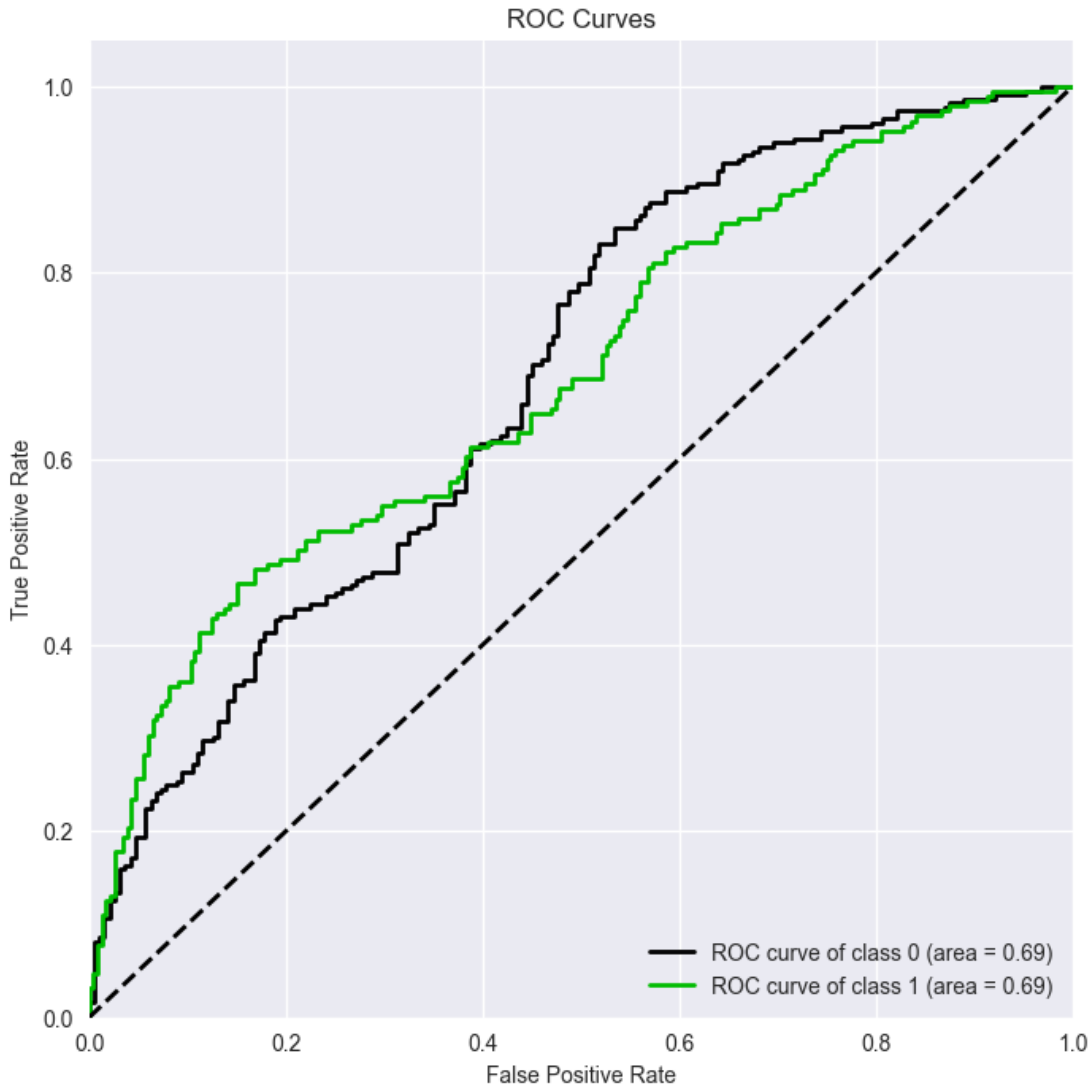
```
    # Tune activation functions
    hp_activation1 = hp.Choice(name = 'activation', values = ['relu', 'elu'], ordered = False)
    hp_activation2 = hp.Choice(name = 'activation', values = ['relu', 'elu'], ordered = False)
    hp_activation3 = hp.Choice(name = 'activation', values = ['relu', 'elu'], ordered = False)

    model.add(LSTM(hp_units1, input_shape=(seqlen, numfeat), activation=hp_activation1, return_sequences=True, name='LSTM1'))
    model.add(Dropout(hp_dropout1, name='Drouput1'))

    model.add(LSTM(hp_units2, activation = hp_activation2, return_sequences=True, name='LSTM2'))
    model.add(Dropout(hp_dropout2, name='Drouput2'))

    model.add(LSTM(hp_units3, activation = hp_activation3, return_sequences=False, name='LSTM3'))

    model.add(Dense(units=1, activation='sigmoid', name='Output'))

    # specify optimizer separately (preferred method))
    opt = Adam(lr=hp_learning_rate, epsilon=1e-08, decay=0.0)

    # model compilation - 'binary_crossentropy' - 'accuracy' - BinaryAccuracy(name='accuracy', threshold=0.5)
    model.compile(optimizer=opt,
                  loss=BinaryCrossentropy(),
                  metrics=['accuracy',
                           Precision(),
                           Recall()])

    return model

# initialize an early stopping callback to prevent the model from
# overfitting/spending too much time training with minimal gains
callback1 = [EarlyStopping(patience=5, monitor='loss', mode='min', verbose=1, restore_best_weights=True),

TensorBoard(log_dir="./tensorboard/hblogs_model_1")]

callback2 = [EarlyStopping(patience=5, monitor='loss', mode='min', verbose=1, restore_best_weights=True),

TensorBoard(log_dir="./tensorboard/hblogs_model_2")]
2024-01-19 19:54:54.230582: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2024-01-19 19:54:54.236866: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
2024-01-19 19:54:54.302256: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.
2024-01-19 19:54:54.331618: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2024-01-19 19:54:54.331643: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
2024-01-19 19:54:54.340301: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.

# HyperBand algorithm from keras tuner
hbtuner = kt.Hyperband(
    build_model,
    objective="val_accuracy",
    max_epochs=5,
    hyperband_iterations=15,
    directory="./keras",
    project_name="hbtrail",
    overwrite=True
)

# launch tuning process
hbtuner.search(g, epochs=50, validation_data=g_, callbacks=callback1, class_weight = class_weight, shuffle=False)
Trial 150 Complete [00h 00m 12s]
val_accuracy: 0.4515366554260254

Best val_accuracy So Far: 0.6406619548797607
Total elapsed time: 00h 27m 53s
INFO:tensorflow:Oracle triggered exit

# display the best hyperparameter values for the model based on the defined objective function
best_hbhp = hbtuner.get_best_hyperparameters()[0]
print(best_hbhp.values)
{'units1': 12, 'units2': 8, 'units3': 20, 'Dropout_rate': 0.3000000000000000004, 'learning_rate': 0.01, 'activation': 'elu', 'tuner/epochs': 5, 'tuner/initial_epoch': 0, 'tuner/bracket': 0, 'tuner/round': 0}

# display tuning results summary
hbtuner.results_summary()
Results summary
Results in ./keras/hbtrail
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
units1: 12
units2: 8
units3: 20
Dropout_rate: 0.30000000000000004
learning_rate: 0.01
activation: elu
tuner/epochs: 5
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.6406619548797607
Trial summary
Hyperparameters:
units1: 8
units2: 24
units3: 20
Dropout_rate: 0.2
learning_rate: 0.01
activation: elu
tuner/epochs: 5
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.6335697174072266
```

```
Trial summary
.
.
.
.
Score: 0.5957446694374084
Trial summary
Hyperparameters:
units1: 8
units2: 12
units3: 28
Dropout_rate: 0.4
learning_rate: 0.001
activation: elu
tuner/epochs: 2
tuner/initial_epoch: 0
tuner/bracket: 1
tuner/round: 0
Score: 0.5910165309906006

tuned_callback1 = [EarlyStopping(patience=5, monitor='loss', mode='min', verbose=1, restore_best_weights=True),
        TensorBoard(log_dir="./tensorboard/hypermodel")]
2024-01-19 14:36:28.814854: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2024-01-19 14:36:28.814885: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
2024-01-19 14:36:28.815507: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.
We will now create a hypermodel based on the best parameters that we received from our tuning process.

hypermodel = hbtuner.hypermodel.build(best_hbhp)

hypermodel.fit(g, epochs=50, verbose=1,callbacks=tuned_callback1,shuffle=False,class_weight=class_weight)
Epoch 1/50
 2/14 [===>.....] - ETA: 2s - loss: 0.7546 - accuracy: 0.5156 - precision: 0.5156 - recall: 1.0000
2024-01-19 14:36:42.183865: I tensorflow/core/profiler/lib/profiler_session.cc:131] Profiler session initializing.
2024-01-19 14:36:42.183892: I tensorflow/core/profiler/lib/profiler_session.cc:146] Profiler session started.
2024-01-19 14:36:42.570617: I tensorflow/core/profiler/lib/profiler_session.cc:66] Profiler session collecting data.
2024-01-19 14:36:42.623200: I tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler session tear down.
2024-01-19 14:36:42.680817: I tensorflow/core/profiler/rpc/client/save_profile.cc:136] Creating directory: ./tensorboard/hypermodel/train/plugins/profile/2024_01_19_14_36_42

2024-01-19 14:36:42.738581: I tensorflow/core/profiler/rpc/client/save_profile.cc:142] Dumped gzipped tool data for trace.json.gz to ./tensorboard/hypermodel/train/plugins/profile/2024_01_19_14_36_42/Aumkars-MacBook-Air.local.trace.json.gz
 4/14 [=======>.....] - ETA: 3s - loss: 0.7595 - accuracy: 0.4922 - precision: 0.4912 - recall: 0.9960
2024-01-19 14:36:42.803722: I tensorflow/core/profiler/rpc/client/save_profile.cc:136] Creating directory: ./tensorboard/hypermodel/train/plugins/profile/2024_01_19_14_36_42
2024-01-19 14:36:42.804310: I tensorflow/core/profiler/rpc/client/save_profile.cc:142] Dumped gzipped tool data for memory_profile.json.gz to ./tensorboard/hypermodel/train/plugins/profile/2024_01_19_14_36_42/Aumkars-MacBook-Air.local.memory_profile.json.gz
2024-01-19 14:36:42.813261: I tensorflow/core/profiler/rpc/client/capture_profile.cc:251] Creating directory: ./tensorboard/hypermodel/train/plugins/profile/2024_01_19_14_36_42
Dumped tool data for xplane.pb to ./tensorboard/hypermodel/train/plugins/profile/2024_01_19_14_36_42/Aumkars-MacBook-Air.local.xplane.pb
Dumped tool data for overview_page.pb to ./tensorboard/hypermodel/train/plugins/profile/2024_01_19_14_36_42/Aumkars-MacBook-Air.local.overview_page.pb
Dumped tool data for input_pipeline.pb to ./tensorboard/hypermodel/train/plugins/profile/2024_01_19_14_36_42/Aumkars-MacBook-Air.local.input_pipeline.pb
Dumped tool data for tensorflow_stats.pb to ./tensorboard/hypermodel/train/plugins/profile/2024_01_19_14_36_42/Aumkars-MacBook-Air.local.tensorflow_stats.pb
Dumped tool data for kernel_stats.pb to ./tensorboard/hypermodel/train/plugins/profile/2024_01_19_14_36_42/Aumkars-MacBook-Air.local.kernel_stats.pb

14/14 [==============================] - 7s 109ms/step - loss: 0.7579 - accuracy: 0.4885 - precision: 0.4492 - recall: 0.4902
Epoch 2/50
14/14 [==============================] - 1s 53ms/step - loss: 0.7564 - accuracy: 0.4728 - precision: 0.4491 - recall: 0.6402
Epoch 3/50
14/14 [==============================] - 1s 68ms/step - loss: 0.7563 - accuracy: 0.4627 - precision: 0.4432 - recall: 0.6512
Epoch 4/50
14/14 [==============================] - 1s 61ms/step - loss: 0.7559 - accuracy: 0.5065 - precision: 0.4720 - recall: 0.6061
.
.
.
14/14 [==============================] - 1s 74ms/step - loss: 0.7215 - accuracy: 0.6350 - precision: 0.6118 - recall: 0.5671
Epoch 22/50
14/14 [==============================] - 1s 55ms/step - loss: 0.7243 - accuracy: 0.5952 - precision: 0.5489 - recall: 0.6780
Restoring model weights from the end of the best epoch.
Epoch 00022: early stopping

<keras.callbacks.History at 0x7fa6ae81bb50>

eval_results = hypermodel.evaluate(g_, verbose=0)

print(f"test loss: {eval_results[0]}, test accuracy: {eval_results[1]}")
test loss: 0.6307883262634277, test accuracy: 0.6619385480880737

get_model_results(hypermodel, g_, y_test, threshold=0.5)
```

Classification Report:

	precision	recall	f1-score	support
0	0.63	0.92	0.75	232
1	0.78	0.35	0.48	191
accuracy			0.66	423
macro avg	0.71	0.63	0.62	423
weighted avg	0.70	0.66	0.63	423



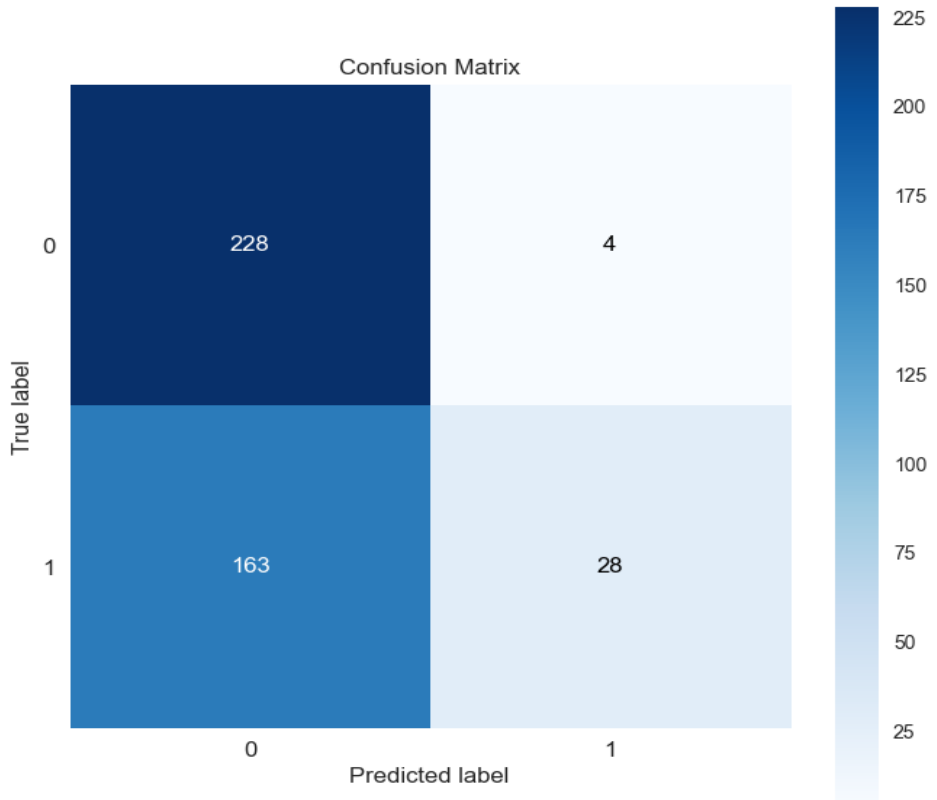


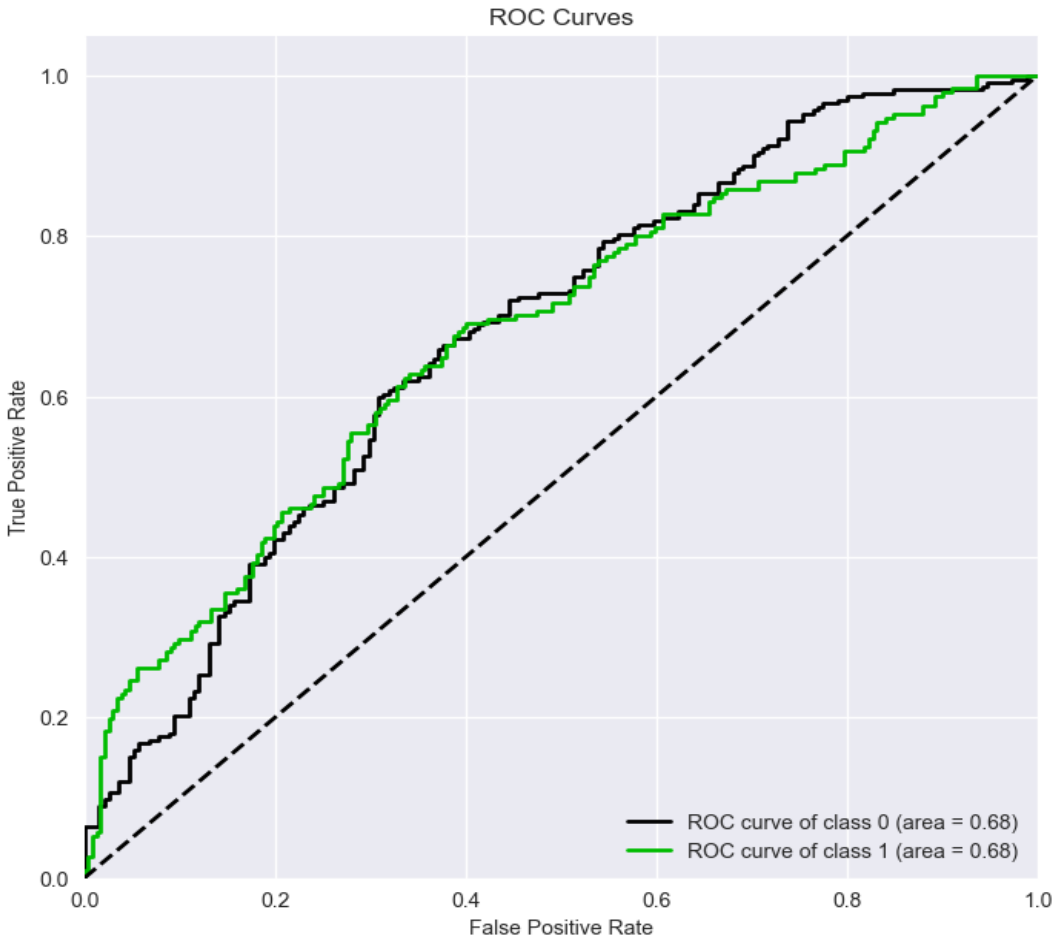
Post tuning model 1, we see that while the accuracy score is roughly similar, the loss has now reduced to 0.6307 as compared to the earlier loss of 0.65780. Thus we can see that indeed the model has now improved.

```
def build_model_2(hp):

    tf.keras.backend.clear_session()

    # instantiate the model
    model = Sequential()

    # Tune the number of units in the layers
    hp_units1 = hp.Int('units1', min_value=4, max_value=32, step=4)

    # Tune the dropout rate
    hp_dropout1 = hp.Float('Dropout_rate', min_value=0, max_value=0.5, step=0.1)

    # Tune the learning rate for the optimizer
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

    # Tune activation functions
    hp_activation1 = hp.Choice(name = 'activation', values = ['relu', 'elu'], ordered = False)

    model.add(LSTM(hp_units1, input_shape=(seqlen, numfeat), activation=hp_activation1, return_sequences=False, name='LSTM1'))
    model.add(Dropout(hp_dropout1, name='Drouput1'))

    model.add(Dense(units=1, activation='sigmoid', name='Output'))

    # specify optimizer separately (preferred method))
    opt = Adam(lr=hp_learning_rate, epsilon=1e-08, decay=0.0)

    # model compilation - 'binary_crossentropy' - 'accuracy' - BinaryAccuracy(name='accuracy', threshold=0.5)
    model.compile(optimizer=opt,
                  loss=BinaryCrossentropy(),
                  metrics=['accuracy',
                           Precision(),
                           Recall()])

    return model

# HyperBand algorithm from keras tuner
hbtuner = kt.Hyperband(
    build_model_2,
    objective="val_accuracy",
    max_epochs=5,
```

```
    hyperband_iterations=15,
    directory="./keras",
    project_name="hbtrail_2",
    overwrite=True
)

# launch tuning process
hbtuner.search(g, epochs=50, validation_data=g_,
callbacks=callback2, class_weight = class_weight,
shuffle=False)
Trial 150 Complete [00h 00m 05s]
val_accuracy: 0.5484633445739746

Best val_accuracy So Far: 0.6690307259559631
Total elapsed time: 00h 15m 30s
INFO:tensorflow:Oracle triggered exit

#display the best hyperparameter values for the model based
on the defined objective function
best_hbhp_2 = hbtuner.get_best_hyperparameters()[0]
print(best_hbhp_2.values)
{'units1': 24, 'Dropout_rate': 0.0, 'learning_rate': 0.01, 'activa
tion': 'relu', 'tuner/epochs': 5, 'tuner/initial_epoch': 2, 'tuner/b
racket': 1, 'tuner/round': 1, 'tuner/trial_id': '9fee8fdcca32c363
f0766b0e54764388'}

# display tuning results summary
hbtuner.results_summary()
Results summary
Results in ./keras/hbtrail_2
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
units1: 24
Dropout_rate: 0.0
learning_rate: 0.01
activation: relu
tuner/epochs: 5
tuner/initial_epoch: 2
tuner/bracket: 1
tuner/round: 1
tuner/trial_id: 9fee8fdcca32c363f0766b0e54764388
Score: 0.6690307259559631
Trial summary
.
.
.
Score: 0.6477541327476501
Trial summary
Hyperparameters:
units1: 28
Dropout_rate: 0.4
learning_rate: 0.001
activation: relu
tuner/epochs: 5
tuner/initial_epoch: 0
tuner/bracket: 0
tuner/round: 0
Score: 0.6477541327476501

hypermodel_2 = hbtuner.hypermodel.build(best_hbhp_2)
tuned_callback2 = [EarlyStopping(patience=5,
monitor='loss', mode='min', verbose=1,
restore_best_weights=True),

TensorBoard(log_dir="./tensorboard/hypermodel_2")]
2024-01-19 14:58:52.501510: I tensorflow/core/profiler/lib/
profiler_session.cc:131] Profiler session initializing.
2024-01-19 14:58:52.501548: I tensorflow/core/profiler/lib/
profiler_session.cc:146] Profiler session started.
2024-01-19 14:58:52.505184: I
tensorflow/core/profiler/lib/profiler_session.cc:164] Profiler
session tear down.

# Model fitting
hypermodel_2.fit(g,
        epochs=50,
        verbose=1,
        callbacks=tuned_callback2,
        shuffle=False,
        class_weight=class_weight)
Epoch 1/50
 2/14 [===>.....] - ETA: 3s - loss: 0.7594 - accu
racy: 0.5352 - precision: 0.5376 - recall: 0.7045
2024-01-19 14:59:01.043537: I tensorflow/core/profiler/lib/
profiler_session.cc:131] Profiler session initializing.
2024-01-19 14:59:01.043581: I tensorflow/core/profiler/lib/
profiler_session.cc:146] Profiler session started.
2024-01-19 14:59:01.525041: I tensorflow/core/profiler/lib/
profiler_session.cc:66] Profiler session collecting data.
2024-01-19 14:59:01.600023: I tensorflow/core/profiler/lib/
profiler_session.cc:164] Profiler session tear down.
2024-01-19 14:59:01.676329: I tensorflow/core/profiler/rpc/
client/save_profile.cc:136] Creating directory: ./tensorboard/
hypermodel_2/train/plugins/profile/2024_01_19_14_59_01

2024-01-19 14:59:01.695821: I tensorflow/core/profiler/rpc/
client/save_profile.cc:142] Dumped gzipped tool data for tra
ce.json.gz to ./tensorboard/hypermodel_2/train/plugins/profil
e/2024_01_19_14_59_01/Aumkars-MacBook-Air.local.trac
e.json.gz
 8/14 [================>.....] - ETA: 0s - loss: 0.7
603 - accuracy: 0.4893 - precision: 0.4728 - recall: 0.7407
2024-01-19 14:59:01.753253: I tensorflow/core/profiler/rpc/
client/save_profile.cc:136] Creating directory: ./tensorboard/
hypermodel_2/train/plugins/profile/2024_01_19_14_59_01

2024-01-19 14:59:01.769001: I tensorflow/core/profiler/rpc/
client/save_profile.cc:142] Dumped gzipped tool data for me
mory_profile.json.gz to ./tensorboard/hypermodel_2/train/pl
ugins/profile/2024_01_19_14_59_01/Aumkars-MacBook-Ai
r.local.memory_profile.json.gz
2024-01-19 14:59:01.779158: I tensorflow/core/profiler/rpc/
client/capture_profile.cc:251] Creating directory: ./tensorboa
rd/hypermodel_2/train/plugins/profile/2024_01_19_14_59_0
1
Dumped tool data for xplane.pb to ./tensorboard/hypermodel
_2/train/plugins/profile/2024_01_19_14_59_01/Aumkars-M
acBook-Air.local.xplane.pb
Dumped tool data for overview_page.pb to ./tensorboard/hy
permodel_2/train/plugins/profile/2024_01_19_14_59_01/Au
mkars-MacBook-Air.local.overview_page.pb
```

```
Dumped tool data for input_pipeline.pb to ./tensorboard/hyp
ermodel_2/train/plugins/profile/2024_01_19_14_59_01/Au
mkars-MacBook-Air.local.input_pipeline.pb
Dumped tool data for tensorflow_stats.pb to ./tensorboard/hy
permodel_2/train/plugins/profile/2024_01_19_14_59_01/Au
mkars-MacBook-Air.local.tensorflow_stats.pb
Dumped tool data for kernel_stats.pb to ./tensorboard/hyper
model_2/train/plugins/profile/2024_01_19_14_59_01/Aumk
ars-MacBook-Air.local.kernel_stats.pb

14/14 [==============================] - 5s 98ms
/step - loss: 0.7575 - accuracy: 0.5177 - precision: 0.4746 - r
ecall: 0.4451
Epoch 2/50
14/14 [==============================] - 1s 32ms
/step - loss: 0.7559 - accuracy: 0.4615 - precision: 0.4473 - r
ecall: 0.7195
Epoch 3/50
14/14 [==============================] - 1s 41ms
/step - loss: 0.7537 - accuracy: 0.5486 - precision: 0.5060 - r
ecall: 0.8268
Epoch 4/50
14/14 [==============================] - 1s 33ms
/step - loss: 0.7536 - accuracy: 0.4964 - precision: 0.4672 - r
ecall: 0.6695
.
.
.
Epoch 12/50
14/14 [==============================] - 0s 28ms
/step - loss: 0.7555 - accuracy: 0.5379 - precision: 0.4971 - r
ecall: 0.3183
Epoch 13/50
14/14 [==============================] - 0s 30ms
/step - loss: 0.7517 - accuracy: 0.4857 - precision: 0.4714 - r
ecall: 0.9646
Epoch 14/50
14/14 [==============================] - 0s 27ms
/step - loss: 0.7508 - accuracy: 0.5587 - precision: 0.5132 - r
ecall: 0.8049
Restoring model weights from the end of the best epoch.
Epoch 00014: early stopping


<keras.callbacks.History at 0x7fa6abd75640>

# evaluate the model
eval_results_2 = hypermodel_2.evaluate(g_, verbose=0)

print(f"test loss: {eval_results_2[0]}, test accuracy:
{eval_results_2[1]}")
test loss: 0.6619855761528015, test accuracy: 0.6052009463
310242

get_model_results(hypermodel_2, g_, y_test, threshold=0.5)
Classification Report:

              precision    recall  f1-score   support

           0       0.58      0.98      0.73       232
           1       0.88      0.15      0.25       191

    accuracy                           0.61       423
   macro avg       0.73      0.56      0.49       423
weighted avg       0.71      0.61      0.51       423
```






```
import pyfolio as pf
```

Trading Strategy and Backtesting

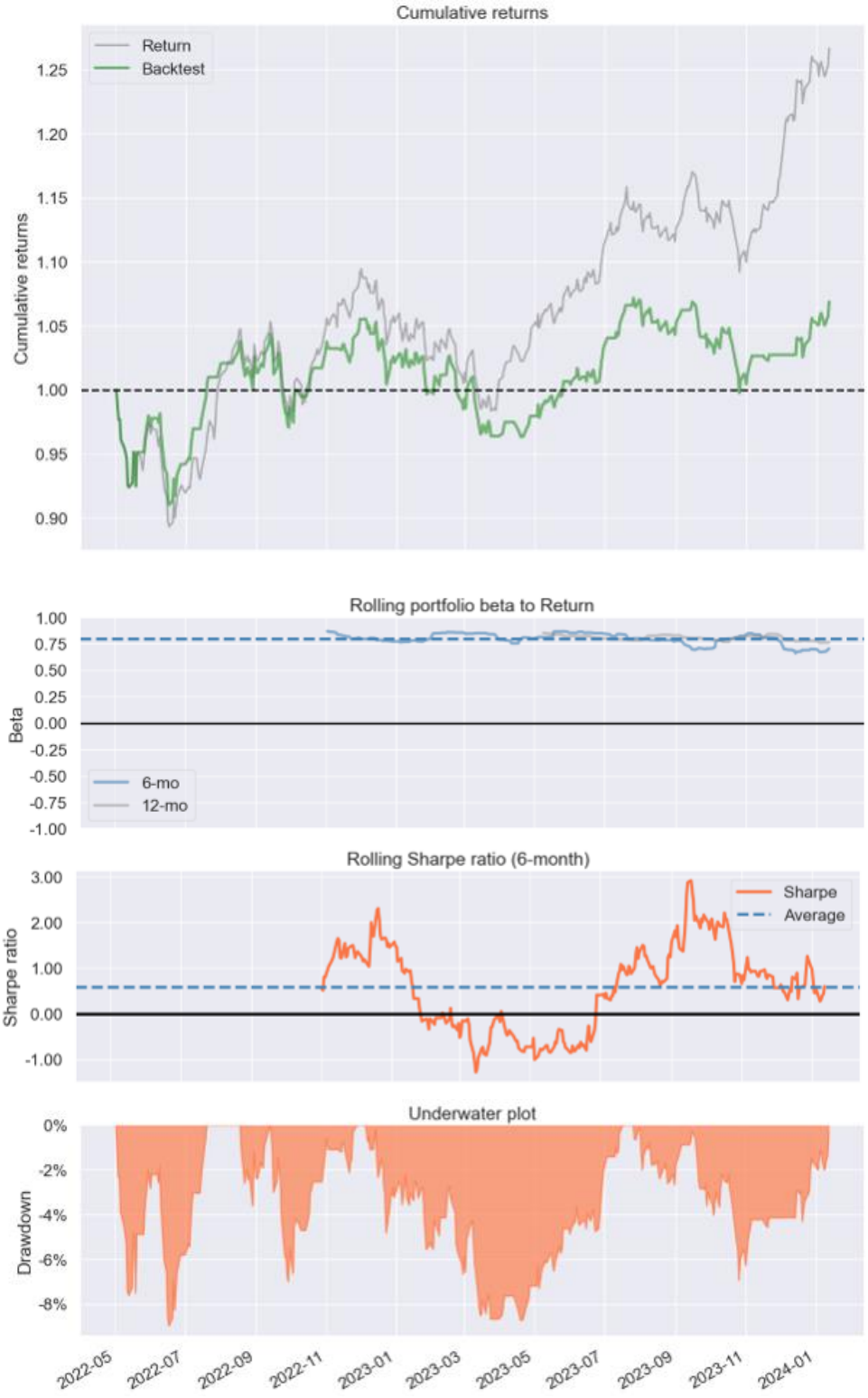
We will now use our best model to make predictions on the entire dataset to see how the algortihm will perform based on a simple trading strategy.

The function uses the model we built to predict the signals for the trading strategy. It compares the predicted values with the threshold and assigns 1 if the prediction is above the threshold, otherwise 0. It calculates the strategy returns by multiplying the daily returns with the lagged signal values.

```
def get_backtest_results(keras_model, g_, df:pd.DataFrame, threshold=0.5, show_display=False):
    """
    Calculates backtest results using a keras model, a test generator dataset and the daily returns DataFrame (df),
    to get the trading strategy information. The threshold for the trading strategy can be varied to find the optimal
    threshold for a model.
    """
    Y_pred = np.where(keras_model.predict(g_, verbose=False) > threshold, 1, 0)
    df_2 = pd.DataFrame(df.iloc[-1*Y_pred.shape[0]:])
    df_2['Signal'] = Y_pred
    df_2['Return'] = np.log(df_2['Adj Close']).diff().fillna(0)
    df_2['Strategy'] = df_2['Return'] * df_2['Signal'].shift(1).fillna(0)
    display(pf.create_simple_tear_sheet(df_2['Strategy'], benchmark_rets=df_2['Return']))

    return df_2

model_df_results = get_backtest_results(hypermodel, g_, df, threshold=0.3, show_display=True)
```

Start date	02-05-2022	
End date	12-01-2024	
Total months	20	
	Backtest	
Annual return	4.00%	
Cumulative returns	6.90%	
Annual volatility	11.10%	
Sharpe ratio	0.41	
Calmar ratio	0.45	
Stability	0.34	
Max drawdown	-9.00%	
Omega ratio	1.08	
Sortino ratio	0.58	
Skew	-0.22	
Kurtosis	2.81	
Tail ratio	0.86	
Daily value at risk	-1.40%	
Alpha	-0.07	
Beta	0.82	

None



For a relatively low threshold of 0.3, we see that the model provided an annual return of just 4% over a 20 months period. If we were to reduce the threshold then we would see that the return scales up to about 15%. However, if the threshold was to be made more stringent of say 0.6 then we would see that the strategy yields major losses over the given period.

Conclusion

In this paper, we tried to understand if the LSTM model is able to predict a price movement of a stock index. For this, we generated a set of features gotten from technical analysis indicators which were then reduced down to have only the relevant ones. Using the best features, we constructed three models with different architectures to see if we are able to predict the movement correctly. We optimized the best two models by tuning our parameters. The model that had the best scores was then selected to run our simple trading strategy wherein, depending on the choice of threshold, we saw contrasting results for the given model.

I believe that the model is of course still in it's nascent stages and would require a lot more attention in all aspects of it's creation (features, model building, tuning, etc.). However, it

was an interesting approach to see how different models can behave, especially sometimes when comparing a shallow model to a multilayered stacked model. This was one of the more interesting aspects of the project for me - to see how the different models behaved despite the different layers of stacking.

References

[1] Advanced Machine Learning 1, Kannan Singaravelu
[2] KNN and SVR for Stock Prediction Python Lab, Kannan Singaravelu
[3] Application of Neural Networks using Tensorflow and Keras Python Lab, Kannan Singaravelu
[4] Stackoverflow