CALIFORNIA STATE UNIVERSITY, SACRAMENTO (CSUS)

COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

SACRAMENTO
STATE

CSC 215-01: ARTIFICIAL INTELLIGENCE

FALL 2020

MINI-PROJECT 2: TIME SERIES FORECASTING USING NN, LSTM AND CNN

Due at 4.00 pm, Wednesday, October 14, 2020

Submitted by:

Harshitha Onkar (Sac State Id: 220262706)
Gargi Prabhugaonkar (Sac State Id: 301111525)

Table of Contents

Problem Statement

To implement fully connected Neural Network (NN), Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN) using time series data to predict stock price and depict the results using RMSE and Regression Lift Chart of test data.

1) To predict [Close] of a day based on the last 7 days data [Open, High, Low, Volume, Close] using full-connected Neural Network model.

2) To predict [Close] of a day based on the last 7 days data [Open, High, Low, Volume, Close] using LSTM model

3) To predict [Close] of a day based on the last 7 days data [Open, High, Low, Volume, Close] using CNN model.

To implement additional features such as:

1) To find the best N value (number of days we should consider in the past) that yields the most accurate model.

2) To use LSTM model to predict stock prices for a continuous future time period (prices in the next five days)

3) To use Keras layer wrappers to create bidirectional LSTM.

Methodology

Dataset

The dataset used for this project is ‘CSC215_P2_Stock_Price.csv’ [1]. It contains 4392 records with 7 features- ‘Date’, ‘Open’, ‘High’, ‘Low’, ‘Close’, ‘Adj_Close’, ‘Volume’.

Helper functions

All common functions used for data preprocessing, data sequencing, visualizing output and plotting functions have been stored separately and imported wherever required.

Task 1: Fully Connected Neural Network

Data preprocessing

Data preprocessing steps for fully connected Neural Network Model:

1) Read data from 'CSC215_P2_Stock_Price.csv'
2) Drop missing values
3) Drop null values
4) Drop columns- 'Date', 'Adj_Close' as per requirement
5) Create Output dataset using data from column 'Close'
6) Encode input dataset with columns- 'Open', 'High', 'Low', 'Close', 'Volume'.
7) Using 'Sequences' function, split data rows into batches of 7 days for prediction
8) For fully connected neural network, create 7*5 input features by merging 7 days rows as single input record and append Output Column
9) Export data to a new .csv file to be used for implementing fully connected neural network and hyperparameter tuning- 'P2_preprocessed_NN.csv'

Dataset changes after each step:

Result after steps	Dataset	x	y
Step 1	(4392,7)		
Step 2	(4392,7)		
Step 3	(4392,7)		
Step 4	(4392,5)		
Step 5		(4392,5)	(4392,)
Step 6		(4392,5)	
Step 7		(4384,7,5)	(4384,)
Step 8		(4384,35)	(4384,)
Step 9	(4384,36)		

Create Sequences

```
import numpy as np

def to_sequences(seq_size, input, target):
    x = []
    y = []

    for i in range(len(target)-seq_size-1):
        window = input[i:(i+seq_size)]
        after_window = target[i+seq_size]
        window = [x for x in window]
        x.append(window)
        y.append(after_window)

    return np.array(x),np.array(y)
```

Fig.1: Code Snippet: Implementation of Step 7

2) Create 7*5 Input features- Merge 7 days rows as single input record and append Output Column

```
[ ] x=x.reshape(x.shape[0],x.shape[1]*x.shape[2])
    x.shape

(4384, 35)

[ ] x_df=pd.DataFrame(data=x,index=None)
    y_df=pd.DataFrame(data=y,index=None)
    result=pd.concat([x_df,y_df], axis=1, ignore_index=True)

    result.columns=[*range(0,36,1)]

    print(x_df.shape)
    print(y_df.shape)
    print(result.shape)

(4384, 35)
(4384, 1)
(4384, 36)
```

Fig.2: Code Snippet: Implementation of Step 7 and Step 8

Implementation

Implementation steps for fully connected Neural Network Model:

1. Import helper functions
2. Load preprocessed data from 'P2_preprocessed_NN.csv' (4384,36)
3. Split data into train set and test using to_xy() function
4. Implement fully connected Neural Network using train set and test set from step 3
5. Use ModelCheckpoint() to save best model and EarlyStopping()
6. Train model using train set and predict using test set.

Data	Train set input-x_train	Test set output-y_train	Test set input-x_test	Test set output-y_test
(records, features)	(3068,35)	(3068,)	(1316,35)	(1316,)

Results for fully connected Neural Network

Model Performance Evaluation for fully connected Neural Network

```
-------- Performance Evalutation for  Neural Networks  --------
-------- Parameters:  {'layers': [50, 10], 'activation': ['ReLu'], 'optimizer': ['adam']}  --------

RMSE         : 1.0285425
MSE          : 1.0578996
R2 score     : 0.9987357900437913
-------- Regression Chart --------
```

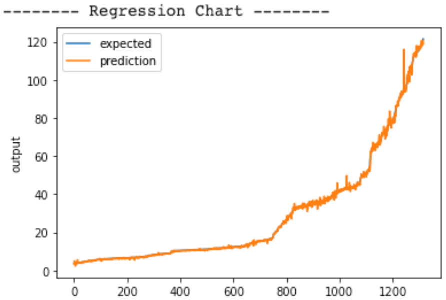


Fig.3: RMSE and Regression Chart for Fully Connected Neural Network

Hyper Parameter Tuning for fully connected Neural Network

The preprocessed data as used in fully connected Neural Network is used for hyperparameter tuning of NN.

Case A: Varying number of hidden layers and neurons

Activation function used: 'relu', optimizer used: 'adam'

# of Hidden Layers	# of Neurons	RMSE
3	100, 10, 50	1.057
3	50, 25, 10	1.092
4	100, 25, 10, 50	1.078

Case B & C: Combination of activation function and optimizers

		adam	sgd	custom adam	custom_sgd
CASE B	**relu**	1.164	28.928	1.474	28.974
	sigmoid	0.994	1.321		
	tanh	1.027	5.327		
CASE C	**relu, sigmoid, tanh**	7.137			

```
*****************************************************************
* RMSE for multiple layers
*****************************************************************
RMSE for layers:  100-10-50  is  1.057745099067688
-----------------------------------------------------------------------
RMSE for layers:  50-25-10  is  1.092740774154663
-----------------------------------------------------------------------
RMSE for layers:  100-25-10-50  is  1.0785863399505615
-----------------------------------------------------------------------
*****************************************************************
* RMSE for activation-optimizer combination
*****************************************************************
RMSE for activation-optimizer:  relu-adam  is  1.1642389297485352
-----------------------------------------------------------------------
RMSE for activation-optimizer:  relu-sgd  is  28.92806053161621
-----------------------------------------------------------------------
RMSE for activation-optimizer:  sigmoid-adam  is  0.9948626756668091
-----------------------------------------------------------------------
RMSE for activation-optimizer:  sigmoid-sgd  is  1.321122407913208
-----------------------------------------------------------------------
RMSE for activation-optimizer:  tanh-adam  is  1.0272386074066162
-----------------------------------------------------------------------
RMSE for activation-optimizer:  tanh-sgd  is  5.327296257019043
-----------------------------------------------------------------------
RMSE for activation-optimizer:  relu-custom_adam  is  1.4747469425201416
-----------------------------------------------------------------------
RMSE for activation-optimizer:  relu-custom_sgd  is  28.974411010742188
-----------------------------------------------------------------------
*****************************************************************
* RMSE for multi activation-optimizer combination
*****************************************************************
RMSE for activation-optimizer:   is  7.137804985046387
-----------------------------------------------------------------------
```

Fig.4: RMSE Output

Task 2: Convolutional Neural Networks (CNN)

Data preprocessing

Data preprocessing steps for CNN and RNN-LSTM:

1) Read data from 'CSC215_P2_Stock_Price.csv'
2) Drop missing values
3) Drop null values
4) Drop columns- 'Date', 'Adj_Close' as per requirement
5) Create Output dataset using data from column 'Close'
6) Normalize input dataset with columns- 'Open', 'High', 'Low', 'Close', 'Volume'.
7) Save preprocessed data to csv for future use.

Dataset changes after each step:

	Step 1	**Step 2 -3**	**Step 4**	**Step 5**	**Step 6**	**Step 7**
Dataframe size	(4392, 7)	(4392, 7)	(4392, 5)	(4392, 6)	(4392, 6)	(4392, 6)

Implementation

Implementation steps for CNN:

1. Import helper functions
2. Load preprocessed data from "P2_Preprocessed_CNN_LSTM.csv"
3. Split the dataframe into input and target sets. (x and y)
4. Convert the input into 4D array using to_sequences and by adding depth dimension
5. Split the dataset into train and test sets.
6. Train and evaluate CNN model using train and test sets.
7. Use ModelCheckpoint() to save best model and EarlyStopping() to stop overfitting.

CASE 1: Each Record as 7 X 5 X 1 (depth of 1)

Data	**Train set input- x_train**	**Test set output- y_train**	**Test set input- x_test**	**Test set output- y_test**
(records, features)	(3068, 7, 5, 1)	(3068,)	(1316, 7, 5, 1)	(1316,)

CASE 2: Each Record as 7 X 1 X 5 (depth of 5)

Data	**Train set input- x_train**	**Test set output- y_train**	**Test set input- x_test**	**Test set output- y_test**
(records, features)	(3068, 7, 1, 5)	(3068,)	(1316, 7, 1, 5)	(1316,)

Results for CNN Model

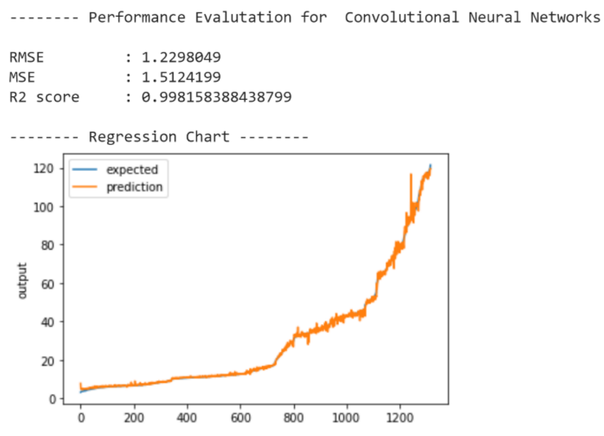


Fig.5: Model Performance for input dim. 7 x 5 x 1

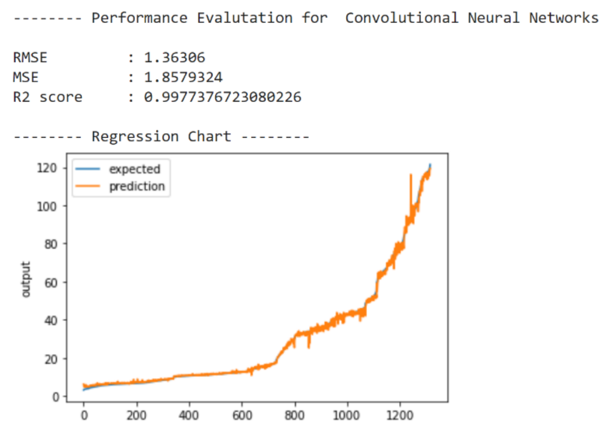


Fig.6: Model Performance for input dim. 7 x 1 x 5

Hyper Parameter Tuning for CNN

Layers and Neuron Count

	Single Conv layer	2 Conv layers	2 Conv layers + Increased Neuron	3 Conv layers
RMSE	1.5728707	1.5689648	1.5920962	1.486823

Optimizer and Activation function

	ReLU	Sigmoid	Tanh
Adam	1.4533412	1.8921036	2.4073963
SGD	8.549669	2.9019604	6.175071

Kernel Count

Single Layer	128	256	512
RMSE	1.3947757	1.3827269	1.4632365

Multi-layer	[32, 64, 128]	[128, 64, 32]	[512, 128, 64]	[1024, 512, 218]
RMSE	1.486823	1.4456707	1.4221526	1.4591217

Kernel Size

	(3,3)	(6,6)	(8,8)
RMSE	1.7468662	1.3800913	1.4663907

Task 3: Recurrent Neural Networks – Long Short-Term Memory (LSTM)

Implementation

Implementation steps for LSTM:

1. Import helper functions
2. Load preprocessed data from
3. Split the dataframe into input and target sets. (x and y)
4. Convert the input into 3D array using to_sequences()
5. Split the dataset into train and test sets.
6. Train and evaluate RNN model using train and test sets.
7. Use ModelCheckpoint() to save best model and EarlyStopping() to stop overfitting.

Data	Train set input-x_train	Test set output-y_train	Test set input-x_test	Test set output-y_test
(records, features)	(3068, 7, 5)	(3068,)	(1316, 7, 5)	(1316,)

Results for LSTM Model

```
-------- Performance Evalutation for  RNN -- with LSTM  -------

RMSE        : 1.174554
MSE         : 1.379577
R2 score    : 0.9983201457586907

-------- Regression Chart --------
```

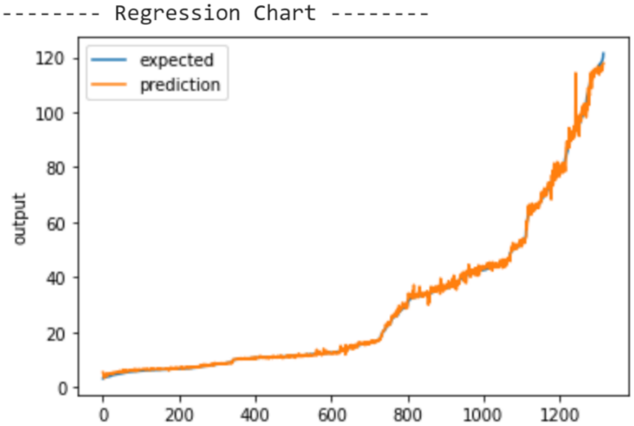


Fig.7: RMSE and Regression lift chart for LSTM

Hyper Parameter Tuning for LSTM

Layers

	Single layer	2 layers	3 layers
RMSE	1.5572765	1.7018794	2.2571971

Neuron Count

Single layer LSTM	64	128	256	512
RMSE	1.6508224	1.6641436	1.435091	1.8889809

Multi-layer	60-30-20	32-64-128	100-50-80	64-128-512	512-128-64
RMSE	2.0348523	1.9946944	2.0874493	2.1298366	1.9981974

Optimizer and Activation function

	ReLU	Sigmoid	Tanh
Adam	1.7163142	0.95701057	1.1522442
SGD	5.6156015	1.9055558	1.3641465

Dropout

	Without Dropout	Regular Dropout	Recurrent Dropout	Both at 20%	Both at 50%
RMSE	1.66804	2.65268	1.5237576	2.908155	6.546654

Additional Features 1: Best N value for stock prediction

To find the best N value (number of days we should consider in the past) that yields the most accurate model.

Implementation

Implementation steps for additional feature 1 using LSTM Model

1. Import helper functions
2. Load preprocessed data from ‘CSC215_P2_Stock_Price.csv’ (4384,36)
3. Preprocess data to drop missing, null values, drop columns, encode columns
4. Perform data sequencing to create input in required dimension
5. Split data into train set and test using to_xy() function
6. List N values= 5, 7, 10, 15, 20, 30, 60
7. Loop through the list N days and compute the best RMSE to create (N, RMSE)
8. Implement LSTM Model using train set and test set from step 5
9. Train model using train set and predict using test set.
10. Compare and print the best N value (number of days we should consider in the past) to yield the most accurate model

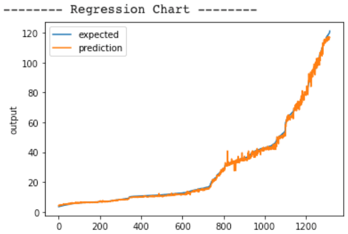
Results

```
RMSE for  5  days: 1.1553571224212646
RMSE for  7  days: 1.7461146116256714
RMSE for  10  days: 1.4214001893997192
RMSE for  15  days: 1.3737393617630005
RMSE for  20  days: 1.3847962617874146
RMSE for  30  days: 1.8228524923324585
RMSE for  60  days: 1.9739160537719727

The best prediction can be done by considering data 5 days in the past with RMSE of 1.1553571224212646
```

Fig.8: Comparison between RMSE for each N value to identify most accurate model

5 Days

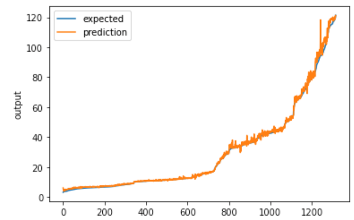
```
Initial shape:  (4392, 5) (4392,)
After sequencing:  (4386, 5, 5)
x_train shape: (3070, 5, 5)
x_test shape: (1316, 5, 5)
y_train shape: (3070,)
y_test shape: (1316,)

RMSE         : 1.3061324
MSE          : 1.705982
R2 score     : 0.9980170781195058
-------- Regression Chart --------
```




```
-----------------------------------------------
RMSE for  5  days is: 1.306132435798645
-----------------------------------------------
```

7 Days

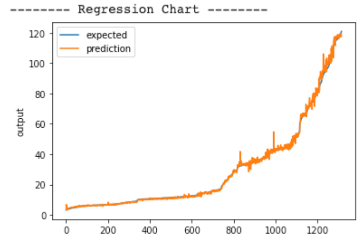
```
Initial shape:  (4392, 5) (4392,)
After sequencing:  (4384, 7, 5)
x_train shape: (3068, 7, 5)
x_test shape: (1316, 7, 5)
y_train shape: (3068,)
y_test shape: (1316,)

RMSE         : 1.6533864
MSE          : 2.7336864
R2 score     : 0.9966713025971644
-------- Regression Chart --------
```




```
-----------------------------------------------
RMSE for  7  days is: 1.6533863544464111
-----------------------------------------------
```

10 Days

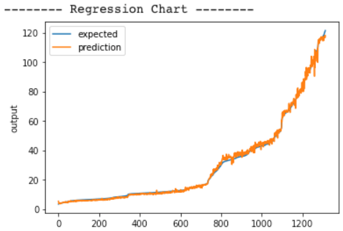
```
Initial shape:  (4392, 5) (4392,)
After sequencing:  (4381, 10, 5)
x_train shape: (3066, 10, 5)
x_test shape: (1315, 10, 5)
y_train shape: (3066,)
y_test shape: (1315,)

RMSE         : 1.4081146
MSE          : 1.9827865
R2 score     : 0.997648783828431
-------- Regression Chart --------
```




```
-----------------------------------------------
RMSE for  10  days is: 1.4081145524978638
-----------------------------------------------
```

15 Days

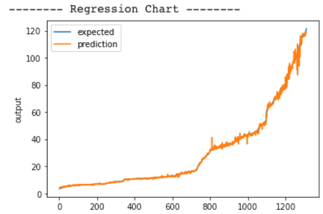
```
Initial shape:  (4392, 5) (4392,)
After sequencing:  (4376, 15, 5)
x_train shape: (3063, 15, 5)
x_test shape: (1313, 15, 5)
y_train shape: (3063,)
y_test shape: (1313,)

RMSE         : 1.4901156
MSE          : 2.2204447
R2 score     : 0.997440137970676
-------- Regression Chart --------
```




```
-----------------------------------------------
RMSE for  15  days is: 1.4901156425476074
-----------------------------------------------
```

20 Days

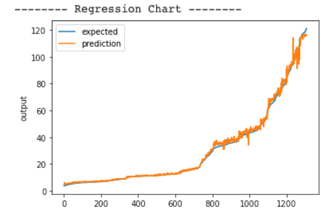
```
Initial shape:  (4392, 5) (4392,)
After sequencing:  (4371, 20, 5)
x_train shape: (3059, 20, 5)
x_test shape: (1312, 20, 5)
y_train shape: (3059,)
y_test shape: (1312,)

RMSE         : 1.3672363
MSE          : 1.8693349
R2 score     : 0.9978025477543818
-------- Regression Chart --------
```




```
-----------------------------------------------
RMSE for  20  days is: 1.367236256599426З
-----------------------------------------------
```

30 Days

```
Initial shape:  (4392, 5) (4392,)
After sequencing:  (4361, 30, 5)
x_train shape: (3052, 30, 5)
x_test shape: (1309, 30, 5)
y_train shape: (3052,)
y_test shape: (1309,)

RMSE         : 1.8912894
MSE          : 3.5769753
R2 score     : 0.9958382282113949
-------- Regression Chart --------
```

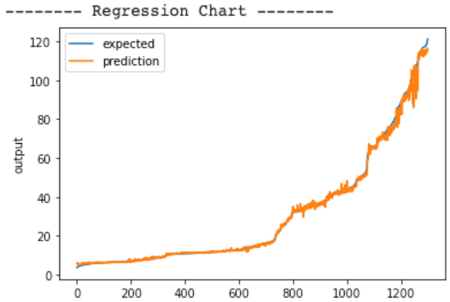



```
-----------------------------------------------
RMSE for  30  days is: 1.8912893533706665
-----------------------------------------------
```

60 Days

```
Initial shape:  (4392, 5) (4392,)
After sequencing:  (4331, 60, 5)
x_train shape: (3031, 60, 5)
x_test shape: (1300, 60, 5)
y_train shape: (3031,)
y_test shape: (1300,)
```

```
RMSE         : 1.7093686
MSE          : 2.921941
R2 score     : 0.9967013850909907
-------- Regression Chart --------
```




```
-----------------------------------------------
RMSE for  60  days is: 1.7093685865402222
-----------------------------------------------
```

Fig.6: RMSE and Regression Chart for each of N values using LSTM Model

Additional Features 2: Multi-output regression model for 5 days prediction

To use LSTM model to predict stock prices for a continuous future time period (prices in the next five days) using Multi-Output Regression Model

Implementation

Implementation steps for additional feature 2 using LSTM Model

1. Import helper functions
2. Load preprocessed data from 'CSC215_P2_Stock_Price.csv' (4384,36)
3. Preprocess data to drop missing, null values, drop columns, encode columns
4. Perform data sequencing to create input in required dimension- output must contain 5 features instead of 1 to predict stock prices for future continuous time period.
5. Split data into train set and test using to_xy() function
6. Implement LSTM Model using train set and test set from step 5
7. Train model using train set and predict using test set- Output contains 5 neurons
8. Evaluate model performance and compare difference between true prices and predicted prices

```
import numpy as np

def this_to_sequences(seq_size, input,target,future_days):
    x = []
    y = []


    for i in range(len(input)-seq_size-1):
        window = input[i:(i+seq_size)]
        window = [x for x in window]
        x.append(window)

    for i in range(len(target)-seq_size-future_days):
        #print(i)
        after_window = target[i+seq_size:i+seq_size+future_days]
        y.append(after_window)

    return np.array(x),np.array(y)
```

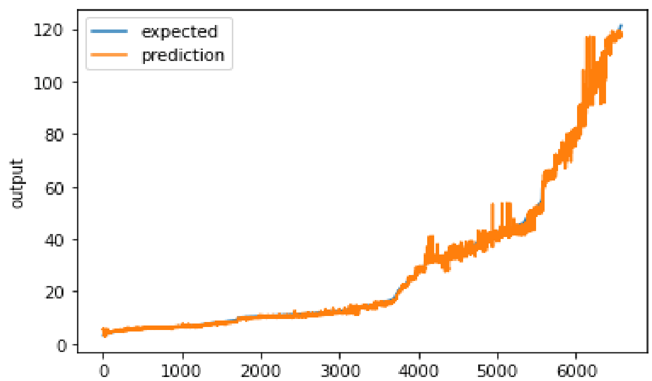
Fig.7: Function to sequence data in required dimension

```
sequence_size 7
Loading data....
dataset: (4392, 7)
dataset after dropping NA rows:  (4392, 7)
dataset after dropping  ['Date', 'Adj_Close']  columns:  (4392, 5)
(4392, 5) (4392,)
Initial shape:  (4392, 5) (4392,)
Shape of x and y before sequencing (4392, 5) (4392,)
Shape of x and y after sequencing (4384, 7, 5) (4380, 5)
(4384, 7, 5) (4380, 5)
4
Balance data set (4380, 7, 5) (4380, 5)
(4380, 7, 5) (4380, 5)
```

Fig.9: Output- After sequencing and balancing data for model

Results

```
x_train shape: (3066, 7, 5)
x_test shape: (1314, 7, 5)
y_train shape: (3066, 5)
y_test shape: (1314, 5)

Training samples: 3066
Test samples: 1314
Model training begins.....
Time elapsed (hh:mm:ss.ms) 0:00:24.617248
Model prediction begins.....
-------- Performance Evalutation for  LSTM- Additional feature  --------
-------- Parameters:  {'optimizer': ['adam']}  --------

RMSE          : 1.7546054
MSE           : 3.0786402
R2 score      : 0.9963438953309272
-------- Regression Chart --------
```




```
Score (RMSE): 1.7546054124832153


Predicted:
[[ 5.67777    5.6503453  5.6108236  5.716807   5.695375 ]
 [25.036888  25.227993  25.34153   25.27851   25.33774  ]
 [10.184966  10.220897  10.238172  10.262867  10.289916 ]
 [89.73877   90.168564  90.33953   90.366554  90.74834  ]
 [41.150997  41.20307   41.394436  41.378704  41.46545  ]]
Expected:
[[ 5.6225   5.575    5.7425   5.8375   5.94   ]
 [25.685   25.5425  25.7025  26.1275  26.375  ]
 [11.055   11.185   11.2375  11.2325  11.23   ]
 [89.2     85.04    86.86    85.8     84.3    ]
 [43.08    42.71    42.765   43.3     44.1    ]]
```

Fig.10: Output- LSTM to predict stock prices for future time period

Additional Feature 3: Keras Wrapper Layers – Bidirectional LSTM

Data Preprocessing

Data preprocessing for Bidirectional LSTM similar to the LSTM model as in Task 3. Input is converted into 3D array for Bidirectional LSTM model processing. Given below are the data shapes of train and test sets.

	X_train	Y_train	X_test	Y_test
Shape	(3068, 7, 5)	(3068,)	(1316, 7, 5)	(1316,)

Implementation

1. A sequential model is created
2. LSTM layer is wrapped with Bidirectional layer wrapper as shown in Fig.6.
3. For forward and backward Bidirectional model, the LSTM layers are defined separately and then referenced in the Bidirectional wrapper as shown in Fig.7.
4. Model is trained and evaluated on train and test sets

```
print("Model training begins.....")
for i in range(3):
  model = Sequential()
  model.add(Bidirectional(LSTM(128, dropout=0.1, recurrent_dropout=0.1, input_shape=ip_size, return_sequences = True)))
  model.add(Bidirectional(LSTM(64, dropout=0.1, recurrent_dropout=0.1)))
  model.add(Dense(50))
  model.add(Dense(1))

  model.compile(loss='mean_squared_error', optimizer='adam')

  monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=4, verbose=0, mode='auto')
  model.fit(x_train,y_train,validation_data=(x_test,y_test), callbacks=[monitor,checkpoint],verbose=0, epochs=100)
```

Fig.11: Bidirectional wrapper layer

```
print("Model training begins.....")
for i in range(3):
  model = Sequential()
  # defining backward and forward layers
  forward_layer = LSTM(128)
  backward_layer = LSTM(64, activation='relu', go_backwards=True)

  model.add(Bidirectional(forward_layer, backward_layer=backward_layer,input_shape=ip_size, merge_mode="concat"))
  model.add(Dense(64))
  model.add(Dense(1))

  model.compile(loss='mean_squared_error', optimizer='adam')

  monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=4, verbose=0, mode='auto')
  model.fit(x_train,y_train,validation_data=(x_test,y_test), callbacks=[monitor,checkpoint],verbose=0, epochs=100)
```

Fig.12: Bidirectional wrapper layer – Forward and Backward layers

Results

Single Bidirectional wrapper

```
-------- Performance Evalutation for  LSTM - Single Bidirectional layer
-------- Parameters:  {'layers': {'Bidirectional-LSTM': 100}, 'activation

RMSE         : 1.2074665
MSE          : 1.4579754
R2 score     : 0.9982246833614216
-------- Regression Chart --------
```

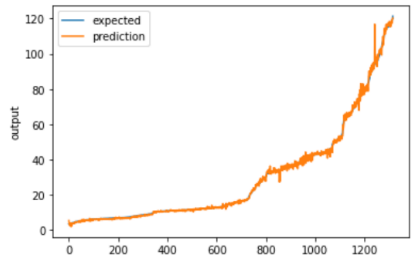


Fig.13: RMSE and Regression lift chart for Single Bidirectional Wrapper

Multiple Bidirectional wrapper

```
-------- Performance Evalutation for  LSTM - Multiple Bidirectional
-------- Parameters:  {'layers': {'Bidirectional-LSTM': [128, 64]},

RMSE         : 1.205203
MSE          : 1.4525144
R2 score     : 0.9982313329356031
-------- Regression Chart --------
```

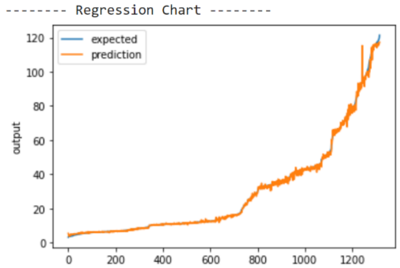


Fig.14: RMSE and Regression lift chart for Multiple Bidirectional Wrapper

Bidirectional wrapper with forward and backward layers

```
-------- Performance Evalutation for  LSTM - Multiple Bidirectional layers  --
-------- Parameters:  {'layers': {'Bidirectional-LSTM': {'forward layer': 128,

RMSE         : 1.3486767
MSE          : 1.8189287
R2 score     : 0.99778516534844
-------- Regression Chart --------
```

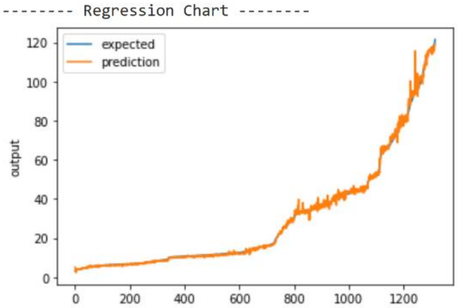


Fig.15: RMSE and Regression lift chart for Bidirectional forward and backward layers

Challenges faced and Takeaways

1) **Visualizing the input as higher dimensional array:** Initially it was difficult to visualize the array as 3D and 4D arrays. But by discussing and understanding to_sequences() implementation we were able to modify it according to the project requirement.
2) **Hyperparameter combinations that don't work:** During Hyperparameter tuning there were some combinations that didn't produce good results or crashed.
3) **Hyperparameters for CNN – Kernel size that results in negative dimensions:** During CNN hyper parameter tuning, certain kernel size configuration led to negative dimension size. The following steps helped to resolve it
   - Adjusting the kernel size according to input size
   - Changing padding from valid to same
   - Referring to proven and tested model hyperparameters of similar type
4) **return_sequences in Multi-layer LSTM:** Understanding return_sequence and return_state parameters helped during multiple LSTM layer stacking.
5) **Multi-output regression model:** Creating the output feature for multi-output regression model was not straightforward. To_sequences() function had to be modified in a way to include continuous future time period
6) **Data Preprocessing:** Performing data preprocessing once and exporting data to csv saved time for model implementation.
7) **Refactoring:** Refactoring code helped to create reusable functions that could be included in helper packages. Although it was time consuming in the initial steps, it saved a lot of time later.
8) **Loops:** Using loops to invoke models for different input parameters allowed to save coding time and space. It also allowed to maintain and modify code easily.

Task Division

Tasks	
Data Preprocessing	Harshitha, Gargi
Fully Connected Neural Networks + Hyperparameter Tuning	Gargi
CNN + Hyperparameter Tuning	Harshitha
LSTM + Hyperparameter Tuning	Harshitha
Additional Feature 1: Best N value for stock prediction	Gargi
Additional Feature 2: Multi-output regression model for 5 days prediction	Gargi
Additional Feature 3: Keras Wrapper layers	Harshitha, Gargi

References

[1] "Stock Price Dataset- Google Drive Link," [Online]. Available: https://drive.google.com/drive/folders/1Bp4_UvWcfZLK2fwiLCpyDcUO9PL9i-Z5.

[2] "Machine Learning Mastery," [Online]. Available: https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/.

[3] "Stack Overflow," [Online]. Available: https://stackoverflow.com.

[4] "RNN," [Online]. Available: https://keras.io/api/layers/recurrent_layers/bidirectional/.

[5] "Bidirectional LSTM," [Online]. Available: https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/.

[6] "Return sequence," [Online]. Available: [2] [3] https://machinelearningmastery.com/return-sequences-and-return-states-for-lstms-in-keras/ .

[7] "Medium," [Online]. Available: https://medium.com/@sanjivgautamofficial/lstm-in-keras-56a59264c0b2.