National University of Computer & Emerging Sciences Karachi Campus

Stock Market Forecaster

Test Report
Software Engineering
Section: 6-F

Group Members:

21K-4554 Bilal Sohail

21K-3309 Yehya Hayati

21K- 3298 Owais Ali Khan

Question #1 (Automated Testing)

Since our project is majorly based upon Machine Learning (ML) models, the test cases can be judged by how close the predicted value is to the actual value of the company stock. Therefore we will be testing the model over a range of values.

Test ID	Test Description	Test Parameters	Test Steps	Test Conditions	Test Case Results
T_1	Testing Google Stock Value with performance metrics taken as MSE.	Google Stock data, and trained model based on LSTM architecture.	Take a trained model and make it predict stock values. Then compare the stock values with the actual stock values on the basis of some performance metrics.	If MSE $\leq$ 40 Then good prediction Else bad prediction, model needs more training	MSE = 2.4692 Excellent Prediction
T_2	Testing Cisco Stock Value with performance metrics taken as MSE.	Cisco Stock data, and trained model based on LSTM architecture.	Take a trained model and make it predict stock values. Then compare the stock values with the actual stock values on the basis of some performance metrics.	If MSE $\leq$ 40 Then good prediction Else bad prediction, model needs more training	MSE = 0.8874 Excellent Prediction
T_3	Testing IBM Stock Value with performance metrics taken as MSE.	IBM Stock data, and trained model based on LSTM architecture.	Take a trained model and make it predict stock values. Then compare the stock values with the actual stock values on the basis of some performance metrics.	If MSE $\leq$ 40 Then good prediction Else bad prediction, model needs more training	MSE = 20.0298 Good Prediction

T_4	Testing Amazon Stock Value with performance metrics taken as MSE.	Amazon Stock data, and trained model based on LSTM architecture.	Take a trained model and make it predict stock values. Then compare the stock values with the actual stock values on the basis of some performance metrics.	If MSE ≤ 40 Then good prediction Else bad prediction, model needs more training	MSE = 5.0138 Excellent Prediction
T_5	Testing Apple Stock Value with performance metrics taken as MSE.	Apple Stock data, and trained model based on LSTM architecture.	Take a trained model and make it predict stock values. Then compare the stock values with the actual stock values on the basis of some performance metrics.	If MSE ≤ 40 Then good prediction Else bad prediction, model needs more training	MSE = 1.2802 Excellent Prediction
T_6	Testing Microsoft Stock Value with performance metrics taken as MSE.	Microsoft Stock data, and trained model based on LSTM architecture.	Take a trained model and make it predict stock values. Then compare the stock values with the actual stock values on the basis of some performance metrics.	If MSE ≤ 40 Then good prediction Else bad prediction, model needs more training	MSE = 4.8892 Excellent Prediction
T_7	Testing General Electrics (GE) Stock Value with performance metrics taken as MSE.	GE Stock data, and trained model based on LSTM architecture.	Take a trained model and make it predict stock values. Then compare the stock values with the actual stock values on the basis of some performance metrics.	If MSE ≤ 40 Then good prediction Else bad prediction, model needs more training	MSE = 44.4038 Bad Prediction, model needs more training

Here is the code used for automated testing (a screenshot has also been attached to see the working of the code):

```
def MSECheckError(calculated_mse: int, error_tolerance: int, index: int):
assert calculated_mse ≤ error_tolerance, "Mean squared error greater than acceptable limits."
print(f"Company:{companyList[index]}, mean squared error: {calculated_mse}")
pred_result={}
MSEs=[]
k=0
for i in stockList:
y_true = scaler[i].inverse_transform(testset[i]["y"]. reshape(-1,1))
y_pred = scaler[i].inverse_transform(regressor.predict(testset[i]["X"]))
MSE = mean_squared_error(y_true, y_pred)
MSECheckError(MSE, 40,k)
k+=1
pred_result_result[i]={}
pred_result[i]"True"] = y_true
pred_result[i]["Pred"] = y_pred
plt.figure(figssize=(14,6))
plt.title("{} with MSE {:10.4f}".format(i,MSE))
plt.plot(y_true)
plt.plot(y_pred)
plt.legend(['y_true', 'y pred'])
```

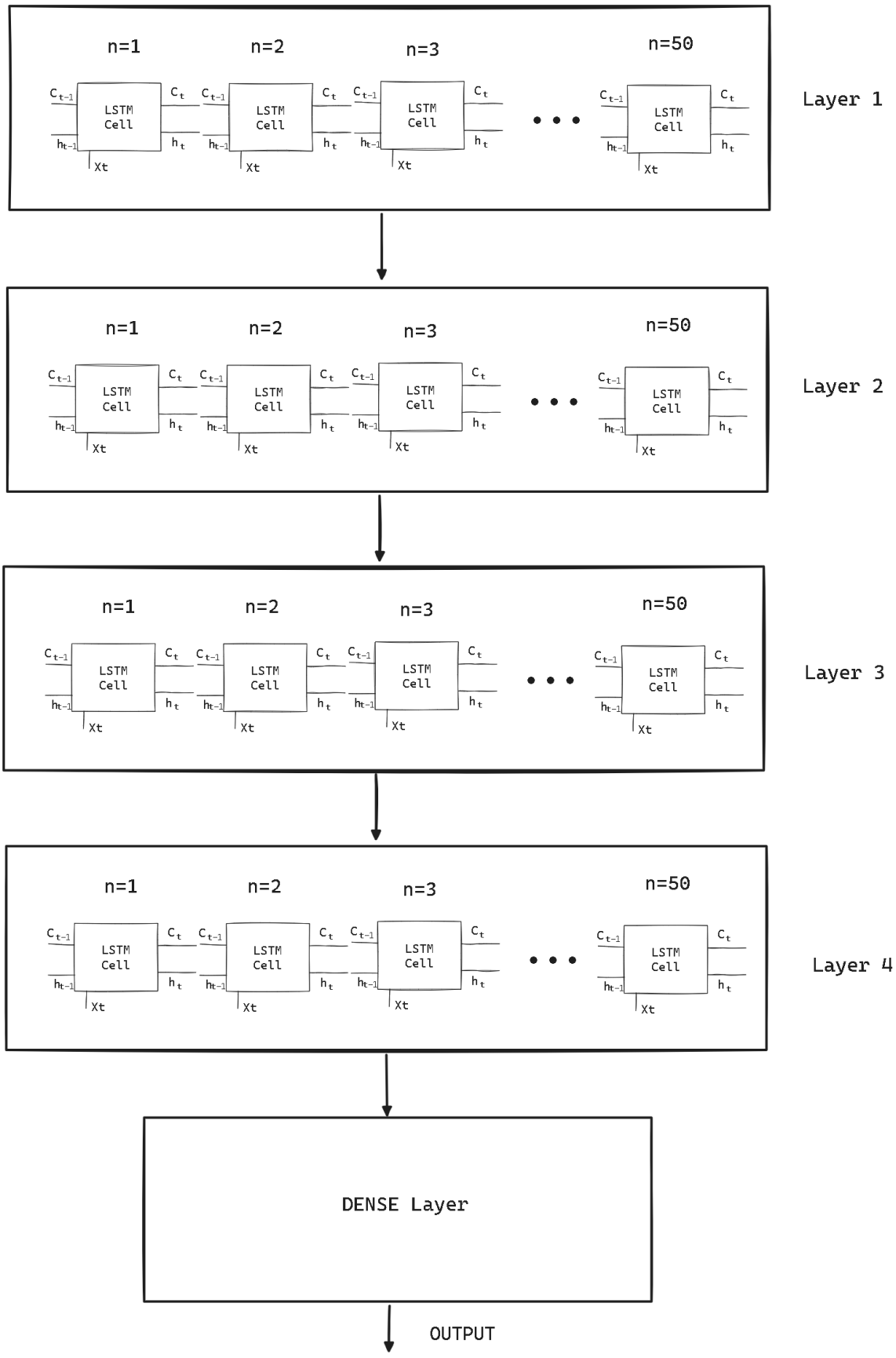
Question #2 (Path Testing)

For path testing we will be considering our training process. Our model is based on a popular deep learning architecture called LSTM.

Long short-term memory (LSTM) units (or blocks) are a building unit for layers of a recurrent neural network (RNN). A RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell is responsible for “remembering" values over arbitrary time intervals; hence the word "memory" in LSTM. Each of the three gates can be thought of as a "conventional" artificial neuron, as in a multi-layer (or feed forward) neural network: that is, they compute an activation (using an activation function) of a weighted sum. Intuitively, they can be thought as regulators of the flow of values that goes through the connections of the LSTM; hence the denotation "gate". There are connections between these gates and the cell.

The expression long short-term refers to the fact that LSTM is a model for the short-term memory which can last for a long period of time. An LSTM is well-suited to classify, process and predict time series given time lags of unknown size and duration between important events. LSTMs were developed to deal with the exploding and vanishing gradient problem when training traditional RNNs.

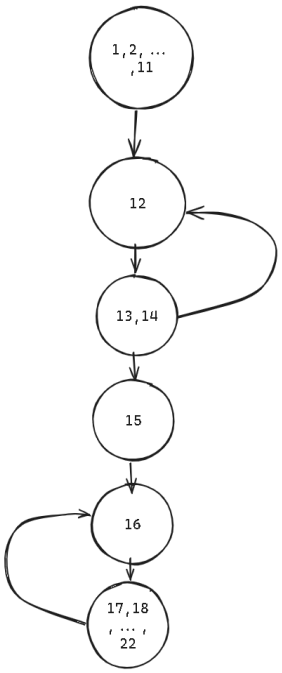
So our approach of constructing the model is to construct a stacked LSTM architecture. We have 4layers of 50 LSTM units each and a dense layer at the end of the network. The specific architecture can be seen below.


n=1
n=2
n=3
n=50
LSTM Cell
Layer 1
Layer 2
Layer 3
Layer 4
DENSE Layer
OUTPUT

Each layer will have a similar structure to this code. We will perform the path testing of the following code.

```
1       regressor = Sequential()
2       regressor.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],1)))
3       regressor.add(Dropout(0.2))
4       regressor.add(LSTM(units=50, return_sequences=True))
5       regressor.add(Dropout(0.2))
6       regressor.add(LSTM(units=50, return_sequences=True))
7       regressor.add(Dropout(0.5))
8       regressor.add(LSTM(units=50))
9       regressor.add(Dropout(0.5))
10      regressor.add(Dense(units=1))
11      regressor.compile(optimizer='rmsprop', loss='mean_squared_error')
12      for i in stockList:
13           print("Fitting to", i)
14           regressor.fit(trainset[i]["X"], trainset[i]["y"], epochs=20, batch_size=100)
15      pred_result = {}
16      for i in stockList:
17           y_true = scaler[i].inverse_transform(testset[i]["y"].reshape(-1,1))
18           y_pred = scaler[i].inverse_transform(regressor.predict(testset[i]["X"]))
19           MSE = mean_squared_error(y_true, y_pred)
20           pred_result[i] = {}
21           pred_result[i]["True"] = y_true
22           pred_result[i]["Pred"] = y_pred
```

This is the following graph.



We will no calculate the cyclomatic complexity of the following graph.

C = E – N + 2

C = 7 – 6 + 2 = 3

Independent paths:

1. 1,12,13,15,16,17
2. 1,12,13,12,13,15,16,17
3. 1,12,13,15,16,17,16,17

Path Testing Test Cases:

Path	Company	Year to Predict	Expected Results (Close Value)
P1	Google	2024 Mar	150.93
P2	Apple	2020 May	79.49
P3	Microsoft	2017 Apr	68.46