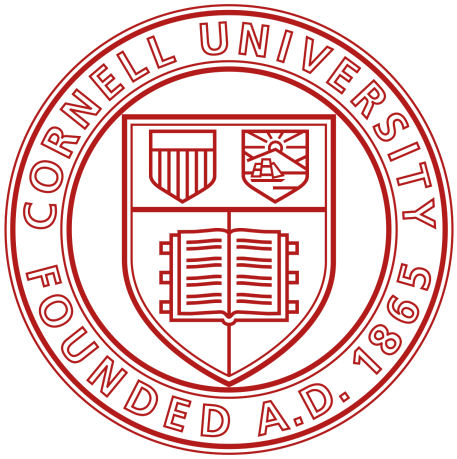


CORNELL UNIVERSITY

DEPARTMENT OF STATISTICAL SCIENCE

Confidence-Based Stock Return Forecasting Model with a Dynamic Time Horizon

Author: Tianjiao Yang (ty366), Nancy Sun (nys5), Shiyao Zhang(sz566), Shiyun Wang (sw983)

Advisor: Dr. Thomas DiCiccio

May 3, 2019

TABLE OF CONTENTS

Executive Summary

MPS Capstone Project

The goal of the project was to build on previous year Cornell MPS projects to predict returns for a universe of stock and financial assets. We did not explore time series analysis in our model research as our stock data failed to meet the stationarity and linearity assumptions required for traditional time series models such as ARIMA. Instead, we mainly investigated and implemented random forest and deep learning models that made use of Recurrent Neural Networks (RNN), specifically LSTM.

In the following section, the report will detail data preparation and cleaning. Then section three and four will discuss and implementation and evaluation of the random forest and long short-term memory (LSTM) models. The models were evaluated using Mean Squared Error (MSE) in addition to cross validation and will be discussed in further detail. Finally, in section five, the report will conclude with model limitations and possible steps to further improve model performance.

1 INTRODUCTION

1.1 Gravity Investments

Gravity Investments is an automated online investing, or RoboAdvisory, company based in Denver, Colorado. The founder, James Damschroder built the company in 2000 around the creation of the robust software application Gsphere, the foundation of Gravity's approach. As Gsphere became interrelated in their clients portfolio strategy architecture and performance optimizations; it naturally progressed to start a Registered Investment Advisory company. A sister company called Gravity Capital Partners was created in 2010. Today both companies are owned by Gravity Investments Inc.

Although there is a widespread acceptance of diversification as an integral investment benefit for portfolio construction, Gsphere was the first genuine method to quantify diversification's existence and effect on a portfolio. As an attestment to its success, Gravity was selected as one of the best Robo Advisors in May 2017.

Gravity's mission is to Bring Life to Diversification. With their pioneering work in the science of portfolio diversification, they empower the industry knowledge and application of True Diversification. So far, approximately $30 billion has been professionally managed with their patented software using Gsphere Diversification Optimization portfolio technology. Their quantitative science and patents for diversification measurement, optimization, and visualization create a unique and compelling experience for investors.

The project was requested by the current CEO of Gravity Investments, James Damschroder. We would like to thank him for introducing us to the application of neural nets in stock market

prediction. We would also like to thank Dr. Thomas Diciccio for his advice and guidance on this project.

2 DATA PREPARATION

Initially, by request of the client, the intention was to use legacy data from previous projects. The legacy data contained financial fundamental features from the Bloomberg terminal and macroeconomic features from the Federal Reserve Bank at St. Louis. But due to poor labeling and structuring of data as well as client concerns to work with more recent information, data was recollected from the Bloomberg terminal.

2.1 Data Collection

The dataset collected from the Bloomberg terminal consists of daily stock returns and other various features of the S&P 500 from the dates 1/1/2009 to 4/17/2019.

2.2 Data Framing

The data was divided into 505 datasets separated by stock. Each dataset included 31 features for each stock. Using client's industry experience, a final 31 out of 100 features are selected. The variables of interest included fundamental financial data such as Current Market Cap, Price Earnings Ratio, Number of Insiders Selling Shares, etc. Response variables used varied by model. The random forest model used Day to Day Total Return (DAY_TO_DAY_TOT_RETURN_NET_DVDS) and the LSTM model used Closing Price 1 Day Ago (PX_CLOSE_1D) as their response variable.

An example dataset of a particular stock is shown in *Figure 1*. In addition, a complete description of all features with their code in Blomberg is listed in the Appendix.

Dates	PX_CLOSE_1D	DAY_TO_DAY_TOT_RETURN_NET_DVDS	CUR_MKT_CAP	LAST_CLOSE_TRR_6MO	VOLATILITY_180D	SHORT_INT_RATIO
1/2/09 0:00	11.1768	3.9028	5716.1668		55.193	
1/5/09 0:00	11.613	3.3867	5909.7562		55.366	
1/6/09 0:00	12.0063	7.9809	6381.4103		56.177	
1/7/09 0:00	12.9645	0.5516	6416.6084		56.053	
1/8/09 0:00	13.036	0.6583	6458.8461		56.033	
1/9/09 0:00	13.1218	0.9264	6518.6828		56.048	
1/12/09 0:00	13.2434	0	6518.6828		55.967	
1/13/09 0:00	13.2434	2.9158	6708.7524		56.096	
1/14/09 0:00	13.6296	-3.2004	6494.0442		56.201	
1/15/09 0:00	13.1934	3.6314	6729.8712		56.262	0.871
1/16/09 0:00	13.6725	2.0397	6867.1437		56.185	

Figure 1: Data set header

2.3 Data Imputation

There were portions of missing data as some companies only became public after the 1/1/2009 and some data were collected monthly or quarterly instead of daily. After consulting with the client, four different imputation methods were used sequentially: forward fill, backward fill, mean fill, zero fill.

3 RANDOM FOREST

3.1 Introduction to Random Forests

The definitions in this subsection are based on the book "An Introduction to Statistical Learning" by James, Witten, Hastie and Tibshirani.[14] Random Forest is a supervised learning algorithm that creates a forest made up of random decision trees. It has been shown to have high prediction accuracy with a large enough number of trees. Another advantage of random forest is that it can be used for both classification and regression problems. In this project, it is used to predict a month ahead of stock returns.

In random forest, random samples are used to build decision trees drawn from data through the bootstrap method. Each tree generates its own predictions and these results would be collected either by a voting method or by averaging the predictions. Predictors are selected at each node.

Bagging, instead, uses exhaustive searches to find the most effective predictors for each node. Unfortunately, bagging does not rule out the influence of any strong predictor and therefore the bagged trees would be highly correlated.

However, this is not the case for building decision trees in Random Forests. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. Each time a tree splits, the candidate variables for creating the split are selected as a sample of *m* predictors drawn from the set of all predictors. The split is allowed to use only one predictor from these *m* randomly selected predictors. This results in a wide diversity that generally results in a better model. Furthermore, as the tree grows larger, more predictors would have a chance to be selected, instead of only exceptionally strong predictors. In practice, the number of predictors in each random sample is a third of all available predictors. Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. We can even make trees more random, by using random thresholds for each feature rather than searching for the best possible thresholds.

3.2 Data and Model Preparation

3.2.1 Imputation

In addition to imputation of missing data by client requests, Random Forest imputation was also examined. It is a machine learning technique which can accommodate nonlinearities and interactions without specifying any particular regression model. The algorithm starts by using the medians of variables to replace missing values. Then, the imputed data is used to train a random forest model to get a proximity matrix, which is used to update the imputed values of missing data. For continuous predictors in our project, the imputed value is the weighted average of non-

missing observations, where the weights are the proximities. Models from both methods of imputation generated similar results.

3.2.2 Parameters in Random Forests

The random forest algorithm was applied in python using RandomForestRegressor in Scikit-Learn. There are several important parameters:

- n_estimators = number of trees in the forest
- max_features = The number of features to consider when looking for the best split. The search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_features.
- max_depth = maximum number of levels in each decision tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples
- min_samples_split = minimum number of samples required to split an internal node
- min_samples_leaf = minimum number of samples allowed in a leaf node
- bootstrap = whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

Initially, the default setting of each parameter were used, but the resulting model did not generate an acceptable accuracy. Using the MSE as an evaluation metric, the parameters were tuned manually. Mean Squared Error (MSE) given by:

$$\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}$$

N = Total Number of Observations

3.3 Accuracy Improvement

3.3.1 Overfitting

At first, the default parameter settings were used, but the prediction accuracy was only 30 percent. This was mostly a result of overfitting. The model performs very well on the training set, but is not able to generalize to a test set. As *figure 2* displays below, the overfitting problem is very obvious. This could be the result of having a large volume of data; using the past 10 years' daily stock returns to predict a month into the future. Showcasing the necessity to tune the hyper parameters for better prediction results.

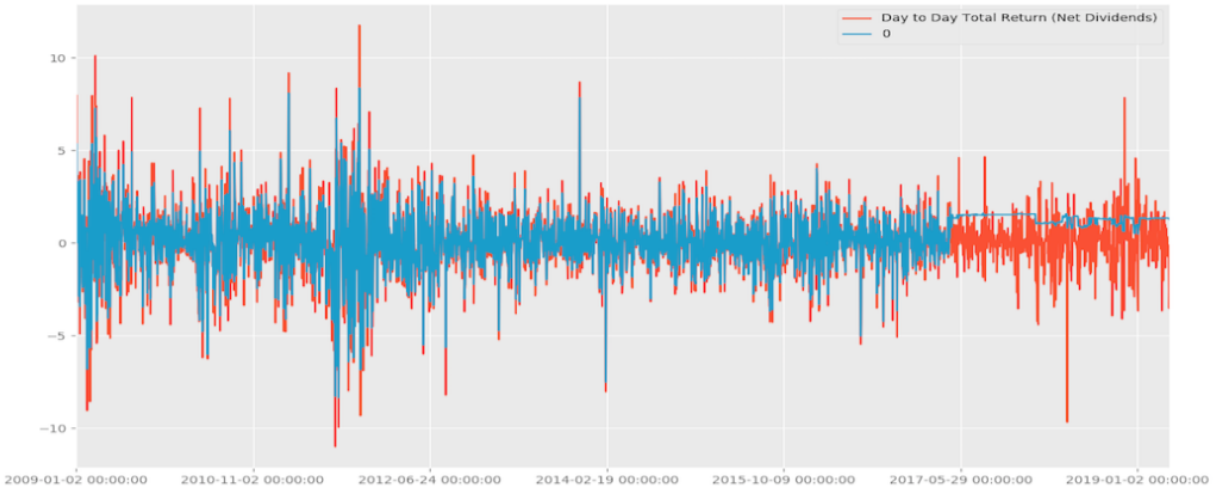


Figure 2

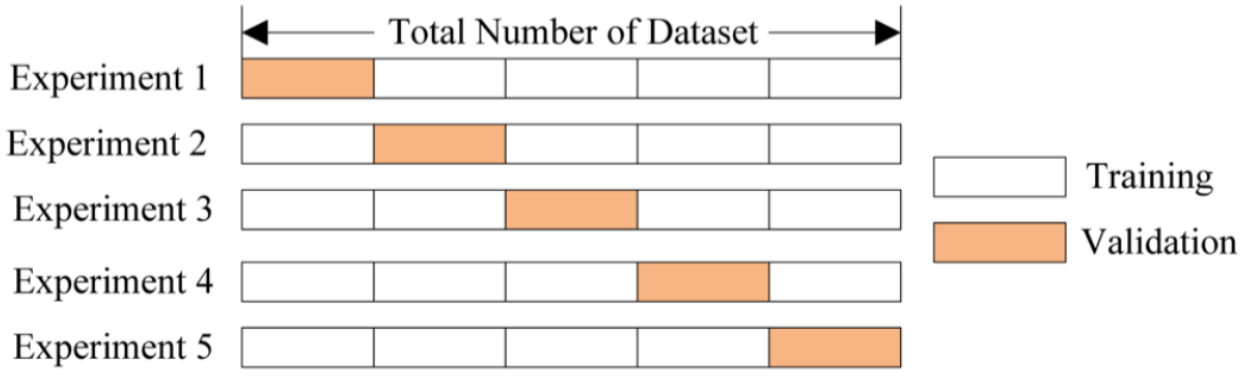
Hyper parameters determine the number of decision trees in the forest, the number of features considered by each tree when splitting a node and so on. Scikit-Learn already implements a set

of sensible default hyper parameters, but these default values can not be guaranteed to be optimal for a specific problem. Intuitively, the best hyper parameters are usually impossible to determine ahead of time.

Hyper parameter tuning relies more on experimental results than theory, and thus many different combinations were evaluated to find the best parameters.

3.3.2 Cross Validation

Cross validation is the standard procedure to avoid overfitting a model. It could be best explained by understanding k-fold cross validation. First, the data is split into a training and a testing set. In K-Fold Cross Validation, the training set is further split into K subsets, called folds. The model is iteratively fit K times, each time only training the data on K-1 of the folds and evaluating on the Kth fold called the validation data. For example, if K = 5, on the first iteration we train on the first four folds and evaluate on the fifth one. The second time we train on the first, second, third, and fifth folds and evaluate on the fourth one. This procedure is repeated for 3 more times, each time evaluating on a different fold. At the end of training, we average the performance of each iteration to come up with final validation metrics for the model.



For hyper parameter tuning, we perform many iterations of the entire K-fold cross validation process, each time using different settings of parameters. By comparing the performance of all the models, we select the best one and train it on the full training set. Then we could evaluate the best model on the test dataset. However, this is a tedious process and may cause a waste of time and loss of efficiency. In other words, every time we want to assess a different set of hyper parameters, we have to split our training data into K folds and train and evaluate for K times. Specifically, if we have 20 sets of hyper parameters and use 5-Fold Cross Validation, there would be 100 training loops in total. Fortunately, model tuning with K-Fold Cross Validation can be automatically implemented in Scikit-Learn by using Scikit-Learn's RandomizedSearchCV method.

3.3.3 Hyper Parameter Tuning

The approach to narrow our search for the best parameters is to evaluate a wide range of values for each hyper parameter. Using Scikit-Learn's RandomizedSearchCV method, we can define a grid of hyperparameter ranges, and randomly sample from the grid, performing K-fold cross validation with each combination of values.

To use RandomizedSearchCV, we first need to create a parameter grid to sample from during fitting. And the grid we use to tune hyper parameters is displayed below:

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
```

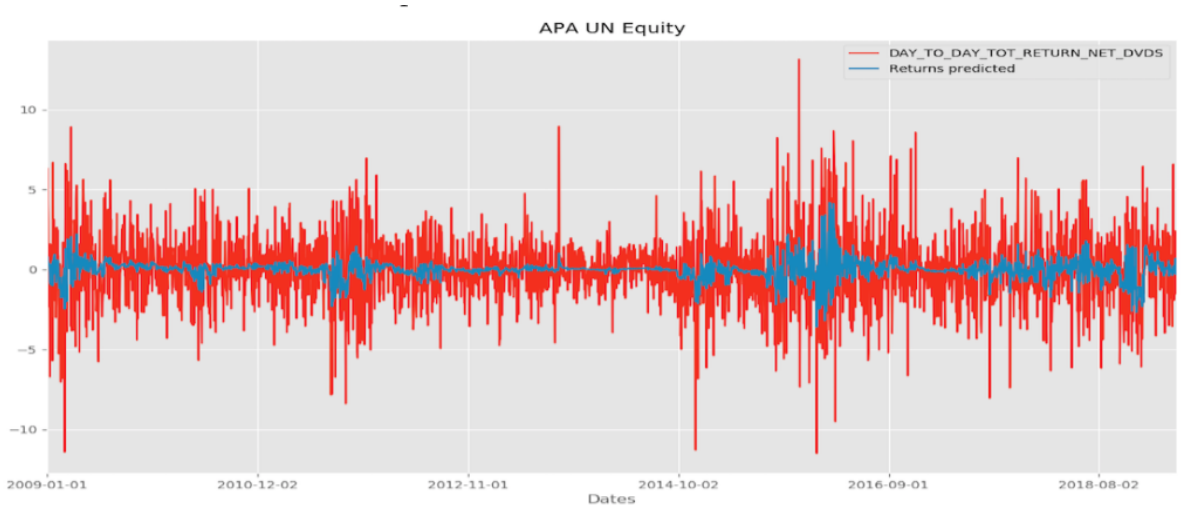
```
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600,
1800, 2000]}
```

On each iteration, the algorithm would choose a different combination of the features. Thus, there are 2 * 12 * 2 * 3 * 3 * 10 = 4320 settings in total. However, we use random search to selecting at random to sample a wide range of values, instead of trying every combination, which reduces the workload to a great extent.

After we run the code and get the best hyper parameters, we need to evaluate the model to see how much improvement we've made. In respect to MSE, we achieve an unspectacular improvement in accuracy shown below. We can further improve our results by using grid search to focus on the most promising hyper parameters ranges found in the random search, but it would require more time to finish.

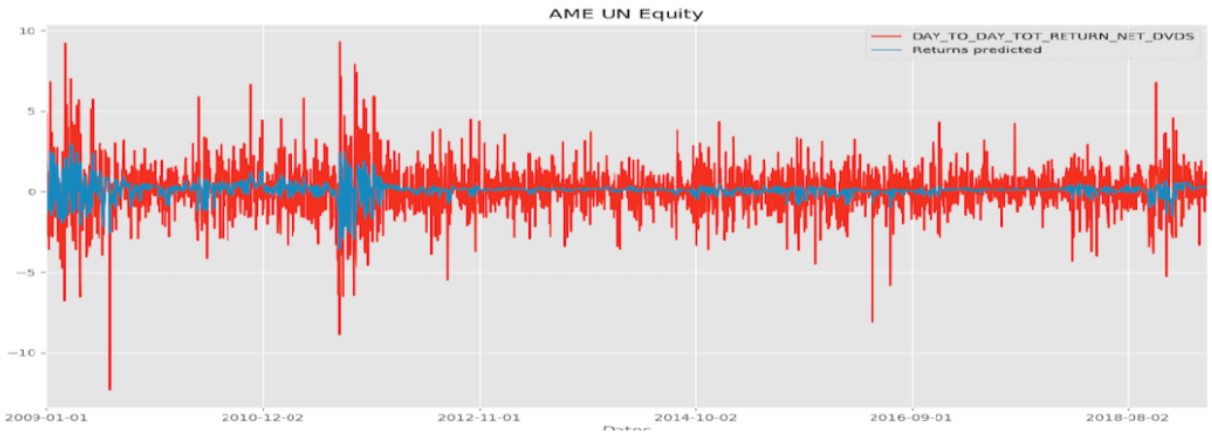
```
>>> base_accuracy = evaluate(base_model, test_features, test_labels)
Model Performance
Average Error: 1.8140 degrees.
Accuracy = 32.35%.
>>> best_random = rf_random.best_estimator_
>>> random_accuracy = evaluate(best_random, test_features, test_labels)
Model Performance
Average Error: 1.4939 degrees.
Accuracy = 78.34%.
>>> print('Improvement of {:0.2f}%.'.format( 100 * (random_accuracy - base_accur
acy) / base_accuracy))
Improvement of 142.19%.
```

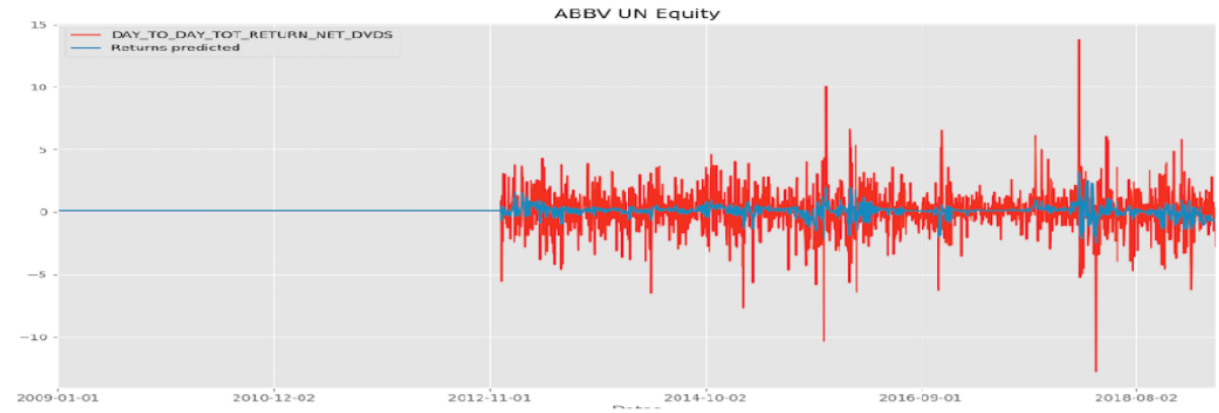
We can see from the prediction figure below that the overfitting has been reduced. The model now performs well in the whole dataset.



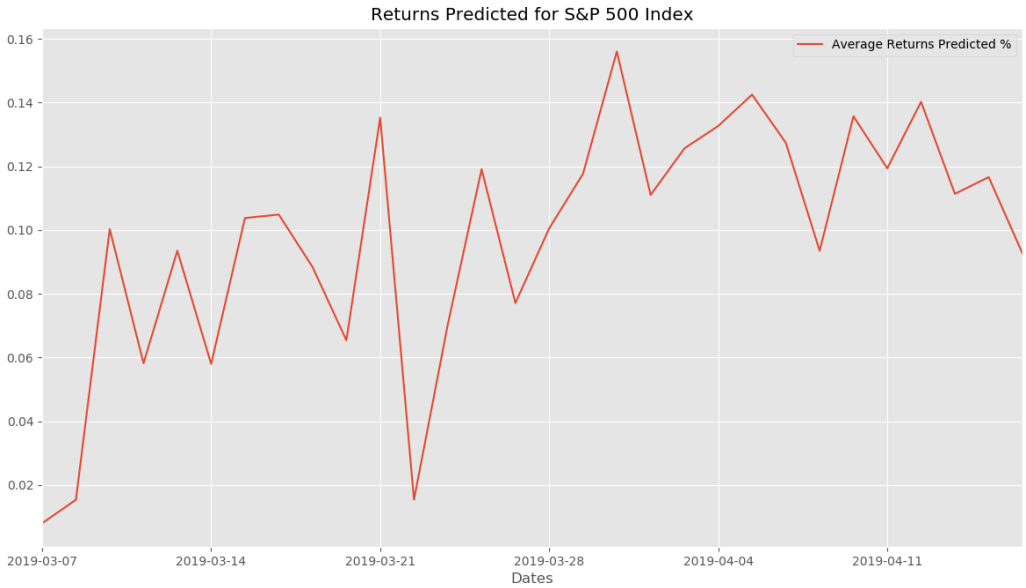
3.4 Results

There were four steps in the process of building the Random Forest model. The first step was to impute the missing values in the raw dataset. Both methods generated similar results. Then, the Random Forest model is fitted based on the whole dataset including stocks information from 01/02/2009 all way to 03/13/2019. There are 30 predictors containing the stock features and some other sentimental variables collected from Bloomberg terminal. The total number of records is 3062 and we use these previous data to make predictions for a month in the future. After that, we tune the hyper parameters by random search to improve the accuracy of the model and avoid overfitting problems. And there two of the predictions for individual stocks are exemplified here.





Finally, we calculated the average returns of 505 stocks for each day and plotted them below. From the figure we generate, we can conclude from a financial investment perspective that the stock market stays quite stable in this month and the total average returns keep positive for these days. In other words, if investing on S&P 500, our client company may get a daily return of approximately 0.10 in this month.



4 LSTM

4.1 Introduction to Neural Nets

Predicting future financial prices or any financial trends has been a challenging while promising field for a long time. When it comes to forecasting future returns on a certain financial asset, it can be defined as predicting future events by analyzing the historical data. This task is innately difficult due to the nonlinear nature of financial time series, and another reason is that the stock price is a response function of a large group of variables or features. This kind of time series data is omnipresent in our lives: speech recognition, handwriting recognition, text translation, weather forecasting, and even psychological signal diagnosis. A stock time series data can be either univariate or multivariate. A univariate time series, in our case, is a sequence of measurements of the same variable over time. Oppositely, a multivariate time series means multiple variables are varying over time. The domain of input data tends to span a group of fields including the historical performance of the company, industry trends, macroeconomic factors, financial fundamentals, and market sentiment, etc. In general, a forecasting problem can be classified in to

three subcategories for stock predictions: short term forecasting, medium term forecasting, and long term forecasting. Eventually the goal would be to achieve long term forecasting with high accuracy, which means predictions beyond 2 years, but in our project, we focused on short-term predictions for now. Many machine learning models have been implemented and tested on time series prediction including but not limited to Regression, Support Vector Machine, Random Forest, Perceptron, and Radial Basis Function (RBF) whereas neural network models have been capturing attention and outperforming due to its ability to capture interdependency in the time series data. Moreover, neural networks are designed to handle non-linear data and consequently are favored over the traditional machine learning techniques. Another reason why it has been extensively utilized in the real world is that a neural network is extremely flexible and can be easily retrained to a new set of data, which is highly suitable to be exploited on the non-linear, constantly evolving and dynamic stock market. J. B. Heaton and his colleagues have investigated deep learning as a tool in the financial world and explained how neural networks could change the "current thinking" in the financial world and its theoretical advantages over traditional predictive models.[1] Lately, many more researches have been done on different types of neural networks and Long-short Term Memory (LSTM) has been proven successful for time series prediction. More specifically, Fischer and Krauss suggest in their paper that Recurrent Neural Network (RNN) and Long-Short Term Memory networks outperform "memory-free" machine learning techniques such as Random Forest and a simple Deep Neural Net (DNN), implying a substantial room for LSTM to be implemented in the financial market prediction domain.[2]

4.2 Context and Basic Theory

Before introducing exact models used in our project, it is necessary to first understand the historical context and basic theory behind Recurrent Neural Networks. It is a variant of a simple

feedback neural network and was initially proposed by John Hopfield in 1982 to capture time dependencies. In 1986, Michael Jordan for the first time applied this innovative structure in neural networks.[3] In 1990, based on Jordan's research, Jeffrey Elman formally introduced the RNN model, which was called Simple Recurrent Network that time, in his research paper and ever since then RNN has been widely applied in many fields including Natural Language Processing, speech recognition, handwriting recognition and other machine learning fields.[4] As a variant of RNN, LSTM was firstly introduced by Hochreiter and Schmidhuber in 1997 and further improved by Alex Graves in 2012. [5&6] RNN was first invented to handle with sequential data. Before it was proposed, in traditional neural network model, the layers were fully connected and there were no connections among different nodes. This feature does well on some tasks such as image recognition but not effectively on many others such as the ones we mentioned beforehand: speech recognition, text recognition, etc. To overcome this obstacle, the neural network is made "recurrent" in the sense that the output of the current sequence comes from the previous sequence. In other words, the nodes have learned how to memorize the past sequences and to communicate with previous nodes. Theoretically, RNN can take input sequences of any length, but disadvantage related with this characteristic is that if the sequences are very long, it will cause a problem of gradient vanishing or exploding since the gradient values will shrink or accumulate exponentially.[7] As a result, it is impractical to use RNN to capture long-term interdependence and LSTM is then introduced to accommodate this situation.

LSTM was originally designed to capture long-term interdependency without memorizing all the information in the past. In some sense, it intends to solve two problems at the same time: reducing the training time and solving the gradient vanishing or exploding problem. In each

hidden layer, LSTM has three additional functions and it solves the gradient vanishing/exploding problem by transforming exponential multiplication to additive linear form. With these new features, LSTM can store both long-term and short-term time-dependent information without experiencing gradient exploding or vanishing, and thus is highly suitable for time series data. Now we have a more detailed explanation below.

4.3 Single-Layer Network

$$y = f(Wx + b)$$

The function above represents a simplest neural network model. The input x is passed along with initial weight W into the activation function f to get the output y.

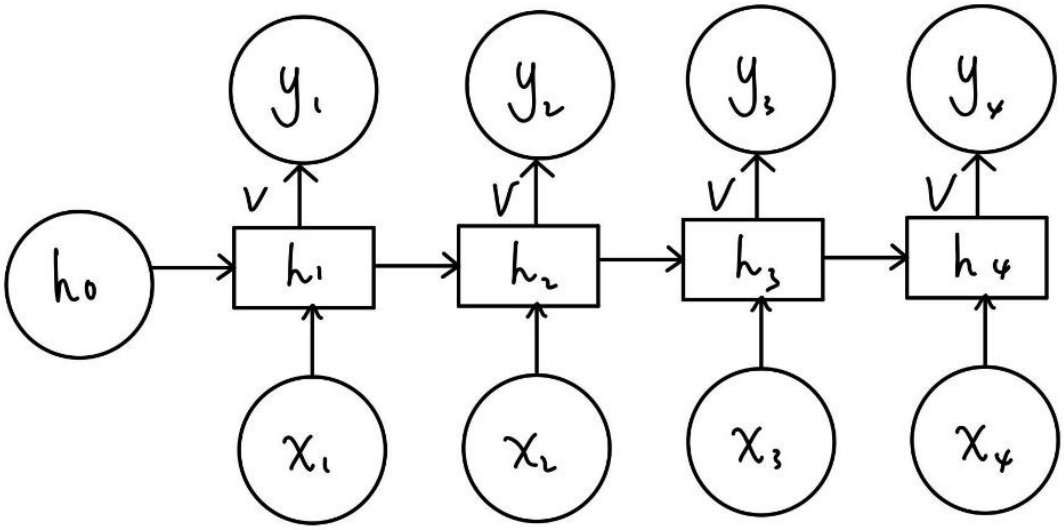
4.4 Classic RNN Structure

In RNN model, the new concept of hidden state, h, is introduced to hold the information from the previous input words. It is defined as:

$$(1)\ h_t = f(W_{hh}h_{t-1} + W_{hx}x_t + b_h)$$

The parameters – W_{hh}, W_{hx} and b_h – should be the same for all the steps, which is a critical trait of RNN model. In the graph below all the circles and squares represent vectors and arrows indicate vector transformation. The graph roughly represents a classical N-to-N RNN structure where y is obtained by the formula:

$$(2)\ y_t = softmax(Vh_t + C)$$

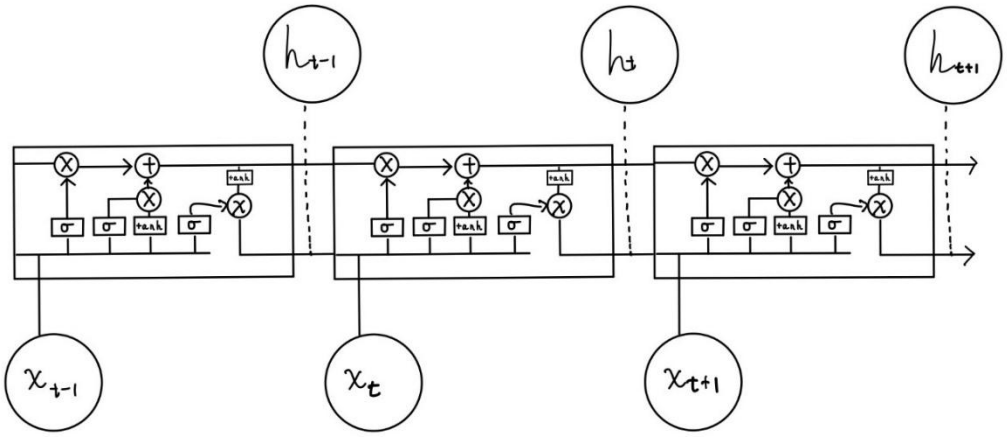


The model can be easily modified to become a 1-to-N, N-to-1, or a N-to-M model if needed. When it comes to N-to-M model, it essentially becomes an Encoder-Decoder model, which is also known as “seq2seq” model and is later adopted in one of our models. How it works for different lengths of input and output is just simply that the encoder will encode the input data into a context vector and then will be passed into another RNN as a decoder to perform the similar transformation explained above.

4.5 LSTM Network

Long Short-term Memory (LSTM) is an advancement of RNN as it has recurrent connections like typical RNN network does but the usual hidden layer is replaced by LSTM memory unit in order to tackle the problem of gradient vanishing or exploding. More specifically, there are three

additional sigmoid functions, so-called “gates,” in each unit, and they control for what to be forgotten, what to be input and what to be output. The following graph is an example of a memory unit in the LSTM model. Internally, there are four gates in total: the forget gate, the input gate, and the candidate gate, and the output gate.



(1) The Forget Gate f_t: in the forgotten gate graph below[1], σ represents sigmoid function here and W_f denotes the weight matrix for this forget gate.

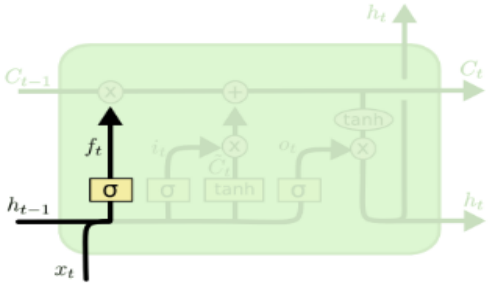


*Figure SEQ Figure * ARABIC 1*

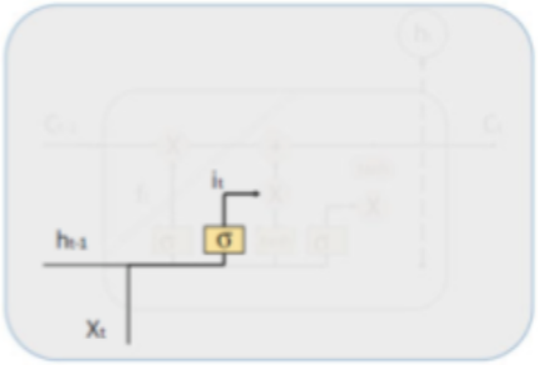
[1] *Fig 1:* https://datascience.stackexchange.com/questions/32217/how-is-the-lstm-rnn-forget-gate-calculated

If the output value f_t is close to 1, then almost all the information is kept while the less is retained if closer to 0. The formula is the following:

$$f_t = \sigma(w_f^T \times h_{t-1} + U_f^T \times x_t + b_f)$$

U_f is the weight matrix between the input layer and the hidden layer.

(2) The Input Gate i_t: the formula is very similar to the forget gate, except that it determines how much of the information newly computed can be passed through.



$$i_t = \sigma\left(w_i{}^T * h_{t-1} + U_i{}^T * x_t + b_i\right)$$

Figure 2

(3) The Candidate Gate: this gate is the one that RNN already has in its internal unit. The activation function here is tanh while in practice it can be replaced with other activation functions such as ReLU and leaky ReLU.

(4) The Output Gate: the following formula, where O_t represents the output and C_t represents the updated memory unit, will determine what to be output after the update has been done

by other gates.

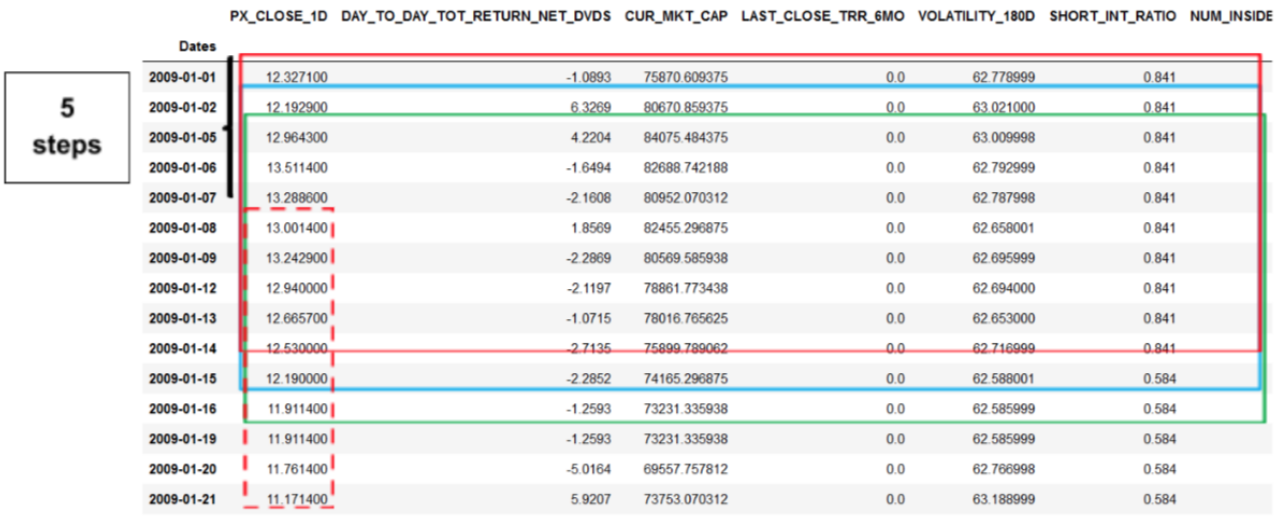
$$O_t = \sigma(w_o^T \times h_{t-1} + U_o^T \times x_t + b_o)$$
$$h_t = O_t \times tanh(C_t)$$

So far, we have introduced a general design of LSTM model in practice. Models can be modified to achieve a variety of purposes under different circumstances.

4.6 Implementations of Different Model Structures

In this project, more than one structure has been experimented and compared to predict future stock prices. Since the final goal is to obtain total return with a dynamic time horizon, it is crucial, firstly, to model out nearly exact stock prices so that returns can be recovered precisely. After browsing through academic literatures, three structures have been carried out under this project framework. First of all, Bontempi et al. points out in their paper that the output can be predicted directly.[8] LSTM has been proved effective and powerful with modeling multivariate time series seamlessly. If using the direct strategy, the model will make predictions at time t+h-1, h= 1,…, H, by framing the time series in a multi-input multi-output form. To do this, before we start modeling, stock data needs to be converted to supervised learning problem. One way to do it is by sliding window approach. In our project, we choose timestep = 1 unanimously for all models but different timesteps can be experimented in the future. The second strategy suggested by Bontempi and other authors is to predict recursively or iteratively, namely, predicting one step ahead and then passing this back to the model to predict another one step after it. The third strategy is to use different parameters for each step, inspired by Cheng and other scholars' results in their paper.[9] In other words, we predict each step separately with different set of parameters.

(1) Direct Method: Before we articulate this method in detail, sliding window method is to be introduced first.



Dates	PX_CLOSE_1D	DAY_TO_DAY_TOT_RETURN_NET_DVDS	CUR_MKT_CAP	LAST_CLOSE_TRR_6MO	VOLATILITY_180D	SHORT_INT_RATIO	NUM_INSIDE
2009-01-01	12.327100	-1.0893	75870.609375	0.0	62.778999	0.841	
2009-01-02	12.192900	6.3269	80670.859375	0.0	63.021000	0.841	
2009-01-05	12.964300	4.2204	84075.484375	0.0	63.009998	0.841	
2009-01-06	13.511400	-1.6494	82688.742188	0.0	62.792999	0.841	
2009-01-07	13.288600	-2.1608	80952.070312	0.0	62.787998	0.841	
2009-01-08	13.001400	1.8569	82455.296875	0.0	62.658001	0.841	
2009-01-09	13.242900	-2.2869	80569.585938	0.0	62.695999	0.841	
2009-01-12	12.940000	-2.1197	78861.773438	0.0	62.694000	0.841	
2009-01-13	12.665700	-1.0715	78016.765625	0.0	62.653000	0.841	
2009-01-14	12.530000	-2.7135	75899.789062	0.0	62.716999	0.841	
2009-01-15	12.190000	-2.2852	74165.296875	0.0	62.588001	0.584	
2009-01-16	11.911400	-1.2593	73231.335938	0.0	62.585999	0.584	
2009-01-19	11.911400	-1.2593	73231.335938	0.0	62.585999	0.584	
2009-01-20	11.761400	-5.0164	69557.757812	0.0	62.766998	0.584	
2009-01-21	11.171400	5.9207	73753.070312	0.0	63.188999	0.584	

The graph shown above is an example of using sliding window size approach to reframe the dataset. Moreover, this example is only showing 1 timestep as forecast sequence. Namely, we are only predicting one step ahead using past 5 steps. The samples will be reshaped as a 3-D form with dimensions [number of samples, number of timesteps, number of features]. More specifically, the initial transformation has been implemented in the “series_to_supervised” function, and afterwards the dimensions of the dataset will become [length of the time series data – input_length, number of features^2 – (number of features -1)]. In our case, if we set series_to_supervised(data, 30, 1), the output from this function will be with size (2685 – 30, 931^2-30) = (2655, 930). Another function called “to_supervised” also has the same functionality but with more flexibility in the input length and output length. Particularly, the timestep can be changed to any number

desired.

After we have converted the data to a supervised form and have split the training and test sets, the model will be fit into out training samples to learn about the past values. The model has been trained on Keras, available on TensorFlow.

(2) Recursive (Iterated) Method: This method is slightly more complex than the first one as we will need to frame our data differently and to add predictions back to our training set. In addition to that, the structure has been made more flexible to be able to have different lengths of the input and output. For example, with modified version of the function, we will be able to use past 10 days of stock data to predict 5 days ahead, in which the length of input and output data do not have to be equal.

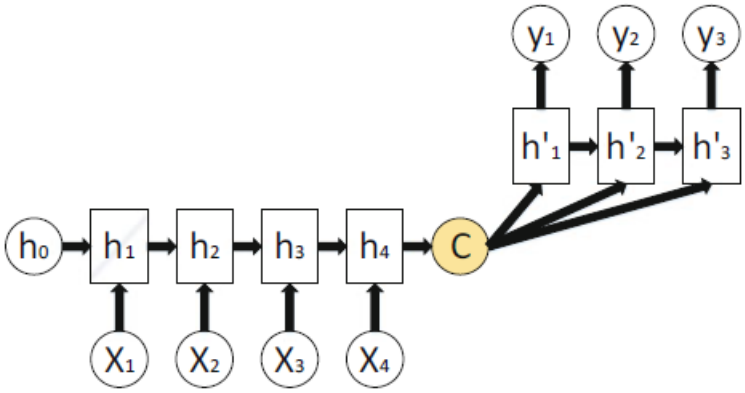
```
# convert history into inputs and outputs
def to_supervised(train, n_input, n_out):
    # flatten data
    data = train.reshape((train.shape[0]*train.shape[1], train.shape[2]))
    X, y = list(), list()
    if(n_input > train.shape[0]*train.shape[1]):
        print("not enough data!")
    # step over the entire data history one time step at a time
    for in_start in range(len(data)):
        # define the end of the input sequence
        in_end = in_start + n_input
        out_end = in_end + n_out
        # ensure we have enough data for this instance
        if out_end < len(data):
            #include all of the features to train the model
            X.append(data[in_start:in_end, :])
            #recording target: the first column
            y.append(data[in_end:out_end, 0])
            #print(data[in_end:out_end, 0])

    train_x_scaler = MinMaxScaler()
    train_y_scaler = MinMaxScaler()

    x_train = np.array(X)
    #print(x_train.shape)
    train_s1,train_s2,train_s3 = x_train.shape[0],x_train.shape[1],x_train.shape[2]
    x_train_scaled = train_x_scaler.fit_transform(x_train.reshape(train_s1*train_s2,train_s3))
    x_train_scaled = x_train_scaled.reshape(train_s1,train_s2,train_s3)
    y_train = np.array(y)
    y_train_scaled = train_y_scaler.fit_transform(y_train)
    print("done to supervised")

    return x_train_scaled, y_train_scaled, train_x_scaler,train_y_scaler
```

After we have converted the data as desired, an Encoder-Decoder model is developed to achieve many-to-many sequence prediction.

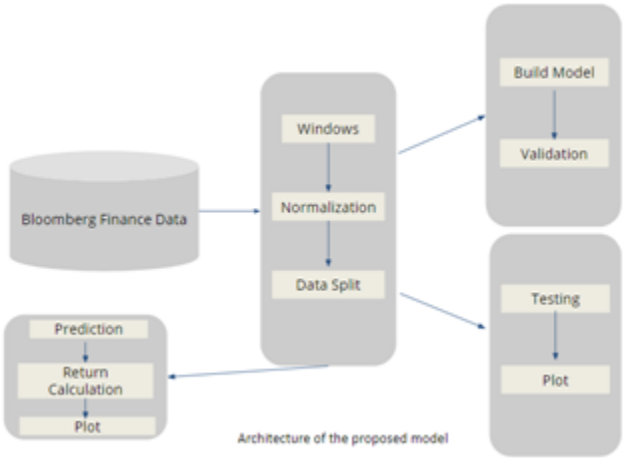


The graph shown above indicates that the Encoder-Decoder model needs the output of the previous sequence to be encoded first and later the decoder will read in this information to make another prediction. It is important to notice here that each input sequence will be encoded separately to generate an internal representation to be decoded later. To achieve this functionality, "RepeatVector(number of days to predict)," "return_sequences=True," and "TimeDistributed" are utilized. Importantly, with "return_sequences=True," the entire sequence, not just the output at the end of the sequence will be returned by the decoder. Then another fully-connected layer will be constructed to interpret each time step in the output sequence before the final output layer. With the "TimeDistributed" wrapper, the decoder will use the same fully connected layer and output layer will be applied to process each time step. This scenario is especially useful for sequence learning when we do not want to flatten out all the output because we want to keep each timestep value separate and we do not want to see any random interactions between different timesteps and features.

(3) Independent Value Prediction Method: this method refers to using different parameter at each time step. In other words, we use a different model for each prediction and the models are independent from each other. This model is initially proposed in the paper of Cheng et al.[9] This method is ideal to avoid overfitting since each model is independent.

However, when time steps increase and sequences become longer, it has difficulty learning the true model unless we tune parameters of each model individually, which is a challenging and computationally expensive model. This is also the reason why it is not performing well in our case since the model needs to learn quite a long sequence with many parameters, including number of hidden layers, batch size, number of epochs, activation functions, and many others available on TensorFlow.

Each model is trained following the procedure presented below. Validation is done by using the parameter already built in Keras - “validation_split” or “validation_data.”

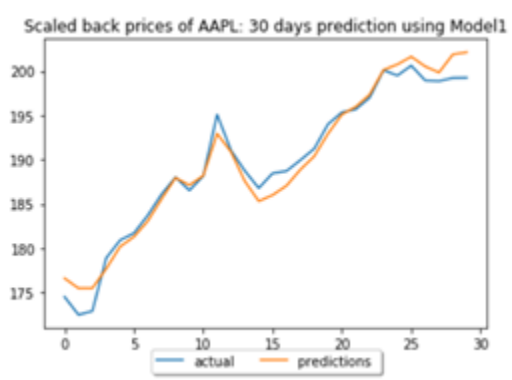


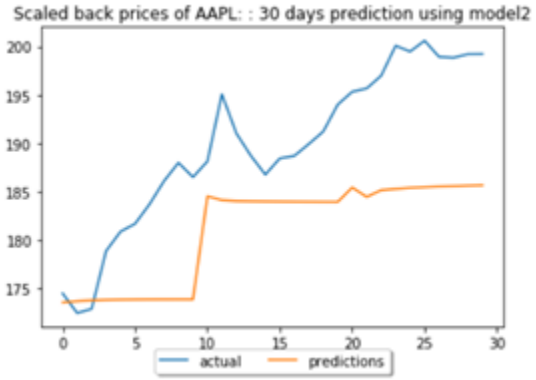
Architecture of the proposed model

4.7 Empirical Results

Three models are first carried out on Apple Inc. stock (AAPL) to see how the models perform on individual stock. After comparing the mean square error (MSE), the best-performing model is then used on 505 stocks (S&P500) to generate 30-day price predictions, which are consequently used to obtain daily net return data. The testing errors for three models are 0.0027364184924711784, 0.07726918458938599 and 0.2258292978008588. It seems from our data that the first model, which is the simplest, outperforms due to the fact that the second one

has error accumulation and that it is overfitting the data. The third model theoretically loses the battle even before we start for the reason mentioned above – we are predicting long sequences.



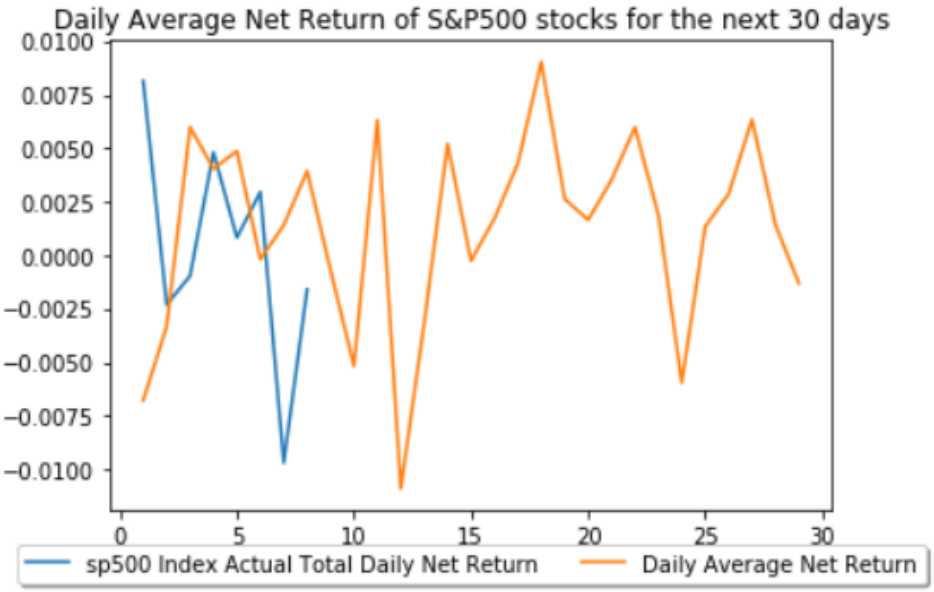


After the first model is claimed to have better performance for our data, the stock data are then input to the model to make full-length predictions. The 30-day prediction results of all 505 stocks are stored in the "model1_pred_table_all.csv," and the training logs that include losses metrics are also saved. Using the net return formula mentioned previously, net returns are thus calculated. The heading of the return table is shown below, and the complete results are saved in "model1_return_table.csv."

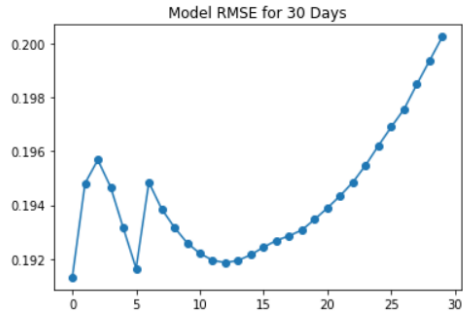
	0	1	2	3	4	5	6	7	8	9	...	20	21	
count	505.000000	505.000000	505.000000	505.000000	505.000000	505.000000	505.000000	505.000000	505.000000	505.000000	...	505.000000	505.000000	505.0000
mean	-0.039179	-0.006772	-0.003320	0.005990	0.004038	0.004869	-0.000187	0.001406	0.003937	-0.000695	...	0.001647	0.003519	0.0059
std	0.070334	0.011301	0.009646	0.008695	0.008946	0.009135	0.010570	0.010585	0.010587	0.008778	...	0.010888	0.010618	0.0097
min	-0.343412	-0.081226	-0.076980	-0.038519	-0.073867	-0.065038	-0.093118	-0.085824	-0.036482	-0.032619	...	-0.047323	-0.041180	-0.0273
25%	-0.079606	-0.011435	-0.004972	0.001784	0.000514	0.001109	-0.002914	-0.002483	-0.001166	-0.004558	...	-0.002512	-0.001973	0.0010
50%	-0.044246	-0.004516	-0.001326	0.004817	0.003207	0.003776	0.000518	0.001762	0.003152	-0.000057	...	0.001474	0.002496	0.0042
75%	-0.005351	-0.000594	0.000562	0.009247	0.007278	0.008319	0.003982	0.006639	0.008683	0.003672	...	0.005795	0.008258	0.0081
max	0.401260	0.047958	0.052491	0.051522	0.042364	0.055579	0.068500	0.043495	0.060062	0.043288	...	0.071852	0.068379	0.0755

8 rows × 30 columns

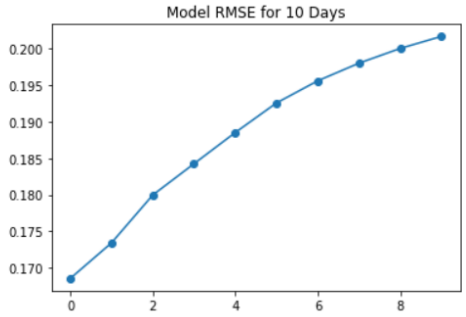
The predictions range from 04/18/2019 to a month later and note here that dividends are not included here and benchmarking these predictions with actual daily net returns would be meaningless. However, an exemplary comparison is still drawn below as a reference and the actual S&P 500 returns are calculated based on the data acquired from Yahoo Finance.



Another significant observation here is that the second model that uses iterative method to predict prices possibly suffer from error accumulation. The RMSE is calculated for cases when we use it to predict 30 days ahead and 10 days ahead.




```
RMSE for each pred is: [0.19131498, 0.19481423, 0.19568332, 0.19467859, 0.19314726, 0.19164751, 0.1948487, 0.19386142
, 0.19317293, 0.19259684, 0.1922033, 0.19194993, 0.19186112, 0.19194587, 0.19215663, 0.1924401, 0.19268821, 0.1928614
7, 0.1930777, 0.19347645, 0.19389944, 0.19435406, 0.19485919, 0.19549526, 0.19622093, 0.19691, 0.19757248, 0.19849361
, 0.19937794, 0.20026618]
```




```
RMSE for each pred is: [0.16856183, 0.17342493, 0.17995599, 0.1842339, 0.18849336, 0.19253819, 0.19556183, 0.19797097
, 0.19998474, 0.20163225]
```

Error accumulation happens when we see the errors keep propagating from the past. More specifically, the result is not getting away from the bias and variance that are generated in the past sequences. In Cheng, Tan, Gao, and Scripps's paper, the bias-variance decomposition is derived to prove that as the prediction step increases, the error accumulation is inevitable no matter what prediction method we use.[9]

5 CONCLUSION: MODEL LIMITATIONS & POTENTIAL

Due to limited time and computational power, there exists limitations in both the random forest and LSTM models. We will discuss some next steps to improve model performance.

5.1 Overall Improvements:

There are a couple of steps that could improve prediction accuracy on both models. First, in our research, it was noted that models could have improved prediction accuracy if high frequency data, such as minute-wise data was used instead of the daily data as we have used in this project. Also the 31 features used in the project were selected based on client expertise. Models could have better accuracy if features were selected using modeling methods in conjunction with client expertise. In addition, stock performance is dependent on a multitude of factors such as news coverage, public opinion on social media, etc. The LSTM model alone is not sufficient to take into account such a variety of predictors. In these cases, natural language processing techniques

can be used to perform sentiment analysis so that it can be included as potential features in a LSTM model, adding potentially valuable data to the model. Lastly, to achieve all of the points mentioned, GPU computing would be essential. So using services such as Amazon Web Services (AWS) would greatly improve project results.

5.2 Random Forest

Results could be improved if it were possible to better tune hyper-parameters because as shown previously, hyper-parameters have drastic results in model accuracy. So instead of using randomized search with cross validation, grid search with cross validation would explore more possible subsets of parameters. Or experimenting with different ratios of training to test data splits.

5.3 Long Short Term Memory

Similar to random forest, parameter tuning plays a big role in model performance. So testing more hyperparameter configurations when training our model could achieve higher prediction accuracy. Also, building hybrid models by combining LSTM with other models could improve prediction. For example, traditional ARIMA and LSTM can be combined to create aARIMA-LSTM hybrid model. The ARIMA model can be used to filter linear tendencies in stock data and the model can therefore account for both linearity and nonlinearity. Residual values are then passed to the LSTM model, which possess long term predictive advantages. According to Hyeong Kyu Choi, ARIMA-LSTM hybrid model outperforms other financial models including full historical model, single-index model, and the multi-group model. Another example of hybrid model is long short-term memory-convolutional neural network (LSTM-CNN) model proposed by Kim T from Ajou University. Different representation of the same data, specifically

in our case, stock financial fundamental data and stock trend charts/news article can be combined to predict stock returns. Instead of using various representation of features about the same stock separately, this hybrid model allows for a combined version of temporal and image features from the same stock data. Last but not least, a TC-LSTM model was introduced by Zhan, X from Huazhong University of Science and Technology in 2018. This novel deep learning model is able to emphasize two crucial aspects of predicting stock returns: the short-term price region and the long-term fluctuation model. LSTM model is used to simulate the short-term trend of the price series. Next, the convolution on the time dimension is used to learn the long-term sequence pattern. Therefore, the advantage of TC-LSTM hybrid model is being capable of obtaining longer stock data dependence and overall change pattern.

5.4 Conclusion

As addressed above, the models explored in this project have demonstrated some proof of concept of with their success in stock market return prediction. But they only serve as a starting point to explore a multitude of cutting edge models limited only by the lack of GPU power to run them. We hope that our project is able to serve as a building step for future project groups to build these models, exploring the limits of machine learning methods applied to the world of stock market prediction.

REFERENCES

1. Heaton, J. B., Nicholas G. Polson, and Jan Hendrik Witte. "Deep learning in finance." *arXiv preprint arXiv:1602.06561*(2016).
2. Fischer, Thomas, and Christopher Krauss. "Deep learning with long short-term memory networks for financial market predictions." *European Journal of Operational Research* 270, no. 2 (2018): 654-669.
3. Jordan, Michael I. "Serial order: A parallel distributed processing approach." In *Advances in psychology*, vol. 121, pp. 471-495. North-Holland, 1997.
4. Nolfi, Stefano, Domenico Parisi, and Jeffrey L. Elman. "Learning and evolution in neural networks." *Adaptive Behavior* 3, no. 1 (1994): 5-28.
5. Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9, no. 8 (1997): 1735-1780.
6. Graves, Alex. "Long short-term memory." In *Supervised sequence labelling with recurrent neural networks*, pp. 37-45. Springer, Berlin, Heidelberg, 2012.
7. Hochreiter, Sepp. "The vanishing gradient problem during learning recurrent neural nets and problem solutions." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6, no. 02 (1998): 107-116.
8. Bontempi, Gianluca, Souhaib Ben Taieb, and Yann-Aël Le Borgne. "Machine learning strategies for time series forecasting." In *European business intelligence summer school*, pp. 62-77. Springer, Berlin, Heidelberg, 2012.
9. Cheng, Haibin, Pang-Ning Tan, Jing Gao, and Jerry Scripps. "Multistep-ahead time series prediction." In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 765-774. Springer, Berlin, Heidelberg, 2006.
10. Pawar, Kriti, Raj Srujan Jalem, and Vivek Tiwari. "Stock Market Price Prediction Using LSTM RNN." In *Emerging Trends in Expert Applications and Security*, pp. 493-503. Springer, Singapore, 2019.
11. Choi, & Kyu, H. (2018, October 01). Stock Price Correlation Coefficient Prediction with ARIMA-LSTM Hybrid Model. Retrieved from https://arxiv.org/abs/1808.01560
12. Kim, T., & Kim, H. Y. (2019). Forecasting stock prices with a feature fusion LSTM-CNN model using different representations of the same data. *Plos One,14*(2). doi:10.1371/journal.pone.0212320

13. Zhan, X., Li, Y., Li, R., Gu, X., Habimana, O., & Wang, H. (2018). Stock Price Prediction Using Time Convolution Long Short-Term Memory Network. *Knowledge Science, Engineering and Management Lecture Notes in Computer Science,*461-468. doi:10.1007/978-3-319-99365-2_41

14. James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Vol. 112. New York: springer, 2013.

15. Brownlee, Jason. "How to Develop LSTM Models for Multi-Step Time Series Forecasting of Household Power Consumption." machinelearningmastery.com. https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/ (accessed April 16th, 2019)

APPENDIX

Variable Description

Bloomberg Code	Description
SHORT_INT_RATIO	Short Interest Ratio
NUM_INSIDERS_SELLING_SHARES	Number of Insiders Selling Shares
NUM_INSIDERS_BUYING_SHARES	Number of Insiders Buying Shares
NUM_INSIDERS_OWNING_SHARES	Number of Insiders Owning Shares
PCT_INSIDER_SHARES_OUT	Percentage Insider Shares Outstanding
PX_TO_CASH_FLOW	Price/Cash Flow
DIVIDEND_12_MONTH_YIELD	Dividend 12 Month Yield
PE_RATIO	Price Earnings Ratio (P/E)
CURR_ENTP_VAL	Enterprise Value
CF_FREE_CASH_FLOW	Free Cash Flow
PX_TO_FREE_CASH_FLOW	Price to Free Cash Flow
PX_TO_BOOK_RATIO	Price to Book Ratio
IS_OPERATING_MARGIN	Operating Margin
PROF_MARGIN	Profit Margin

GROSS_MARGIN	Gross Margin
SALES_REV_TURN	Revenue
NET_REV	Net Revenue
RETURN_ON_ASSET	Return on Assets
RETURN_COM_EQY	Return on Common Equity
GEO_GROW_ROE	Return on Equity - 5 Yr Geometric Growth
GEO_GROW_NET_SALES	Revenue 5 Year CAGR
EPS_GROWTH	EPS - 1 Yr Growth
REVENUE_GROWTH_ADJUSTED_YOY	Revenue Growth Adjusted Year over Year
DILUTED_EPS_5YR_AVG_GR	Diluted EPS - 5 Year Average Growth
EQY_DPS_NET_5YR_GROWTH	Dividend Net 5yr Growth Rate
DEBT_TO_MKT_CAP	Debt To Market Cap Ratio

Python Code

Code for Random Forest

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.ensemble.forest import RandomForestRegressor
from sklearn.preprocessing import Imputer
```

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from pprint import pprint
import glob

def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors / test_labels)
    accuracy = 100-mape
    print('Model Performance')
    print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:0.2f}%.'.format(accuracy))
    return accuracy


df = pd.read_csv('/Users/shiyunwang/Desktop/MPS Project/csv_data_daily_2009_2019/AAP
UN Equity.csv', header=0, index_col=0)
df = df.ffill(axis=0)
df = df.fillna(df.mean())
df = df.fillna(0)
df.isnull().sum()
X = df.drop(['DAY_TO_DAY_TOT_RETURN_NET_DVDS'], axis=1)
y = df['DAY_TO_DAY_TOT_RETURN_NET_DVDS']
n = int(len(df)*0.8)
X_train = X[:n]
y_train = y[:n]
X_test = X[n:]
y_test = y[n:]
# let's get the labels and features in order to run our
# model fitting
train_labels = y_train.as_matrix().reshape(-1,1)
train_features = X_train.as_matrix()
test_labels = y_test.as_matrix().reshape(-1,1)
test_features = X_test.as_matrix()
############## hyperparameter tuning ############
# Number of trees in random forest
```

```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto','sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter =
100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rf_random.fit(train_features, train_labels)

rf_random.best_params_
#{'n_estimators': 1600, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_features': 'sqrt',
# 'max_depth': 10, 'bootstrap': True}
base_model = RandomForestRegressor(n_estimators = 10, random_state = 42)
base_model.fit(train_features, train_labels)
base_accuracy = evaluate(base_model, test_features, test_labels)
best_random = rf_random.best_estimator_
random_accuracy = evaluate(best_random, test_features, test_labels)
print('Improvement of {:0.2f}%.'.format( 100 * (random_accuracy - base_accuracy) /
base_accuracy))
####### Evaluate random search ##########
```

```
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = predictions - test_labels
    mape = 100 * np.mean(errors / test_labels)
    accuracy = abs(mape)
    print('Model Performance')
    print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:0.2f}%.'.format(accuracy))
    return accuracy


# lets set the figure size and color scheme for plots
# personal preference and not needed).
plt.rcParams['figure.figsize']=(25,20)
plt.style.use('ggplot')

files = sorted(glob.glob('/Users/shiyunwang/Desktop/MPS Project/try/*.csv'))
for stock in files:
    df = pd.read_csv(stock, header=0, skiprows=[1], index_col=0)


# fill missing values with mean column values
df = df.ffill(axis=0)
df = df.fillna(df.mean())
df = df.fillna(0)
df.isnull().sum()

X = df.drop(['Day to Day Total Return (Net Dividends)','6 Month Total Return - Last
Close','Free Cash Flow','Operating Margin','Net Revenue','Return on Equity - 5 Yr Geometric
Growth','Revenue Growth Adjusted Year over Year'], axis=1)
y = df['Day to Day Total Return (Net Dividends)']

X_train = X[:2726]
y_train = y[:2726]

X_test = X[2726:]
y_test = y[2726:]

# let's get the labels and features in order to run our
# model fitting
```

```
train_labels = y_train.as_matrix().reshape(-1,1)
train_features = X_train.as_matrix()
test_labels = y_test.as_matrix().reshape(-1,1)
test_features = X_test.as_matrix()

base_model = RandomForestRegressor(n_estimators = 10, random_state = 42)
base_model.fit(train_features, train_labels)
base_accuracy = evaluate(base_model, test_features, test_labels)

best_random = RandomForestRegressor(n_estimators = 10, min_samples_split = 2,
min_samples_leaf = 4,
  max_features = 'sqrt', max_depth = 10, bootstrap = True)
random_accuracy = evaluate(best_random, test_features, test_labels)
# {'n_estimators': 1600, 'min_samples_split': 2, 'min_samples_leaf': 4, 'max_features': 'sqrt',
# 'max_depth': 10, 'bootstrap': True}
print('Improvement of {:0.2f}%.'.format( 100 * (random_accuracy - base_accuracy) /
base_accuracy))

# Now that we've run our models and fit it, let's create
# dataframes to look at the results
X_test_predict=pd.DataFrame(
   best_random.predict(X_test.as_matrix()))

X_train_predict=pd.DataFrame(
   best_random.predict(X_train.as_matrix()))



# combine the training and testing dataframes to visualize
# and compare.
RF_predict = X_train_predict.append(X_test_predict)
RF_predict.set_index(df.index,inplace=True)
RF_predict.columns = ['Prices predicted']
df = pd.concat([df,RF_predict], axis=1)

df[['Day to Day Total Return (Net Dividends)', 'Prices predicted']].plot()
plt.show()

#df['diff']=df['Day to Day Total Return (Net Dividends)'] - df[0]
#df['diff'].plot(kind='bar')
#plt.show()
```

Code for LSTM

```
from core.data_processor import DataLoader
import os
import math
from numpy import array
import numpy as np
import pandas as pd
import datetime
import glob
import random
import datetime as dt
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from numpy import newaxis
####################################
#Data Loading and Preprocessiong
####################################
#load the main excel file that contains S&P500 stock info from Bloomberg
xls = pd.ExcelFile('/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/09_19_fi
nal.xlsx.xlsx');
names=xls.sheet_names
print(names)
#step1:
#export tab sheets to individual csv files
for name in names:
      stock = pd.read_excel(xls,index_col=0,skiprows = range(4),sheet_name =
name)
      name_csv = ("/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/csv_data
_2009_2019/"+name+ ".csv")
      stock.to_csv(name_csv, encoding='utf-8')
      #print(name)
#step2:
```

```
#impute and export again (I exported this seperately from the first step to
be able to change imputation method)
i=0
files = sorted(glob.glob('./csv_data_2009_2019/*.csv'))
for f in files:
      df = pd.read_csv(f,index_col=0)
      df2 = df.iloc[:,1:].astype('float32')
      fill_missing(df2)
      stock = pd.concat([df.iloc[:,0],df2], axis = 1)
      name_csv = ("/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/csv_data
_2009_2019_imputed/"+names[i]+ ".csv")
      stock.to_csv(name_csv, encoding='utf-8')
  i = i+1
#####################################
  #All the data files are saved in the folder/zip file "csv_data_2009_2019")
#####################################

#####################################
#model1_ LSTM_direct_prediction
#####################################

# Importing the libraries
import os
import math
from numpy import array
import numpy as np
import pandas as pd
import datetime
import glob
import random
import datetime as dt
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from numpy import newaxis
from numpy import split
from sklearn import preprocessing
from keras.layers import Dense, Activation, Dropout, LSTM, Embedding,
Flatten, RepeatVector, TimeDistributed
from keras.models import Sequential, load_model, model_from_json
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from matplotlib import pyplot
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
      n_vars = 1 if type(data) is list else data.shape[1]
      df = DataFrame(data)
      cols, names = list(), list()
      # input sequence (t-n, ... t-1)
      for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
      names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    #print(cols)
```

```
      # forecast sequence (t, t+1, ... t+n)
      for i in range(0, n_out):
        cols.append(df.shift(-i))
      if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
      else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
      # put it all together
      agg = concat(cols, axis=1)
      agg.columns = names
      # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
      return agg

cols = ['PX_CLOSE_1D','DAY_TO_DAY_TOT_RETURN_NET_DVDS',
 'CUR_MKT_CAP',
 'LAST_CLOSE_TRR_6MO',
 'VOLATILITY_180D',
 'SHORT_INT_RATIO',
 'NUM_INSIDERS_SELLING_SHARES',
 'NUM_INSIDERS_BUYING_SHARES',
 'NUM_INSIDERS_OWNING_SHARES',
 'PCT_INSIDER_SHARES_OUT',
 'PX_TO_CASH_FLOW',
 'DIVIDEND_12_MONTH_YIELD',
 'PE_RATIO',
 'CURR_ENTP_VAL',
 'CF_FREE_CASH_FLOW',
 'PX_TO_FREE_CASH_FLOW',
 'PX_TO_BOOK_RATIO',
 'IS_OPERATING_MARGIN',
 'PROF_MARGIN',
 'GROSS_MARGIN',
 'SALES_REV_TURN',
 'NET_REV',
 'RETURN_ON_ASSET',
 'RETURN_COM_EQY',
 'GEO_GROW_ROE',
 'GEO_GROW_NET_SALES',
 'EPS_GROWTH',
 'REVENUE_GROWTH_ADJUSTED_YOY',
 'DILUTED_EPS_5YR_AVG_GR',
 'EQY_DPS_NET_5YR_GROWTH',
 'DEBT_TO_MKT_CAP']
#loading data
dataset = read_csv('/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/csv_data
_2009_2019/AAPL UW Equity.csv', header=0, index_col=0)
dataset = dataset.loc[:, cols]
dataset = dataset.ffill(axis=0)
dataset = dataset.bfill(axis=0)
dataset = dataset.fillna(dataset.mean())
dataset = dataset.fillna(0)
dataset = dataset.astype('float32')
ds = dataset.astype('float32')
values = ds.values
```

```
scaler_ds = MinMaxScaler(feature_range=(0, 1))
scaled_ds = scaler_ds.fit_transform(ds)

# frame as supervised learning
reframed_ds = series_to_supervised(scaled_ds, 3, 1)

# drop columns we don't want to predict
# only want to keep predictions of returns here
a = list(range(-30,0,1))
reframed_ds.drop(reframed_ds.columns[[a]], axis=1, inplace=True)


# split into train and test sets
values = reframed_ds.values
n_train_periods = len(values)-30
train = values[:n_train_periods, :]
test = values[n_train_periods:, :]

# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]


# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)


reframed_ds = series_to_supervised(ds,1, 1)
a = list(range(-30,0,1))
#reframed.columns[[a]]
reframed_ds.drop(reframed_ds.columns[[a]], axis=1, inplace=True)
values = reframed_ds.values

n_train_periods = len(values)-30
train = values[:n_train_periods, :]
test = values[n_train_periods:, :]
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]

#scaling
train_x_scaler = MinMaxScaler()
train_y_scaler = MinMaxScaler()
test_x_scaler = MinMaxScaler()
test_y_scaler = MinMaxScaler()

train_x_scaled = train_x_scaler.fit_transform(train_X)
train_y_scaled = train_y_scaler.fit_transform(train_y)

test_x_scaled = test_x_scaler.fit_transform(test_X)
test_y_scaled = test_y_scaler.fit_transform(test_y)


# reshape input to be 3D [samples, timesteps, features]
train_x_scaled = train_x_scaled.reshape((train_x_scaled.shape[0], 1,
train_x_scaled.shape[1]))
```

```
test_x_scaled = test_x_scaled.reshape((test_x_scaled.shape[0], 1,
test_x_scaled.shape[1]))

print(train_x_scaled.shape, train_y_scaled.shape, test_x_scaled.shape,
test_y_scaled.shape)



# design neural network for a single stock
model = Sequential()
model.add(LSTM(200, input_shape=(train_x_scaled.shape[1],
train_x_scaled.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(20,activation='relu'))
model.add(Dense(1))
#model.compile(loss='mse', optimizer='adam')
model.compile(loss='mse', optimizer='adam', metrics=['mse'])
# fit network
#print(test_y.shape)
#print(train_y.shape)
history_aapl = model.fit(train_x_scaled, train_y_scaled, epochs=50,
batch_size=16, validation_split=0.1,verbose=1, shuffle=False)

test_AAPL_loss = model.evaluate(test_x_scaled, test_y_scaled, batch_size =16)
print("Testing MSE for AAPL next 30 days predictions is:",test_AAPL_loss[0])


pred = model.predict(test_x_scaled)
plt.plot( test_y_scaled[:], label="actual")
plt.plot( pred[:,0], label="predictions ")
plt.title("AAPL: 30 days prediction using Model1")
plt.legend( loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True,
shadow=True, ncol=2)
pyplot.show()

test_scaled_back = test_y_scaler.inverse_transform(test_y_scaled)
pred_sclade_back = test_y_scaler.inverse_transform(pred)

#pred = model.predict(test_X)
plt.plot( test_scaled_back, label="actual")
plt.plot( pred_sclade_back, label="predictions ")
plt.legend( loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True,
shadow=True, ncol=2)
plt.title("Scaled back prices of AAPL: 30 days prediction using Model1")
pyplot.show()


#Predicting for Next Month/Year¶

def load_data(file,days_to_predict):
      dataset = read_csv(file, header=0, index_col=0)
      dataset = dataset.loc[:, cols]
      dataset = dataset.ffill(axis=0)
      dataset = dataset.bfill(axis=0)
      dataset = dataset.fillna(dataset.mean())
      dataset = dataset.fillna(0)
      dataset = dataset.astype('float32')
```

```
      reframed_ds = series_to_supervised(dataset, 1, 1)
      a = list(range(-30,0,1))
      #reframed.columns[[a]]
    reframed_ds.drop(reframed_ds.columns[[a]], axis=1, inplace=True)
      values = reframed_ds.values
    n_train_periods = len(values)-days_to_predict
      train = values[:n_train_periods, :]
      test = values[n_train_periods:, :]
      train_X, train_y = train[:, 1:], train[:, 0]
      test_X, test_y = test[:, 1:], test[:, 0]

      #scaling
      train_x_scaler = MinMaxScaler()
      train_y_scaler = MinMaxScaler()
      test_x_scaler = MinMaxScaler()
      test_y_scaler = MinMaxScaler()

      train_x_scaled = train_x_scaler.fit_transform(train_X)
      train_y_scaled = train_y_scaler.fit_transform(train_y)

      test_x_scaled = test_x_scaler.fit_transform(test_X)
      test_y_scaled = test_y_scaler.fit_transform(test_y)


      # reshape input to be 3D [samples, timesteps, features]
      train_x_scaled = train_x_scaled.reshape((train_x_scaled.shape[0], 1,
train_x_scaled.shape[1]))
      test_x_scaled = test_x_scaled.reshape((test_x_scaled.shape[0], 1,
test_x_scaled.shape[1]))

      return(train_x_scaled, train_y_scaled, test_x_scaled,
test_y_scaled,train_x_scaler,train_y_scaler,test_x_scaler, test_y_scaler)


days_to_predict = 30
from keras.callbacks import Callback
#OUTPUT_PATH = ('C:\Users\yangt\OneDrive - Cornell
University\Spring\Project\past\confidence.based.security.forecasting\training
_log\')
from keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLROnPlateau, CSVLogger
csv_logger = CSVLogger(('training.log'), append=True)
import glob
files = sorted(glob.glob('/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/csv_data
_2009_2019/*.csv'))

files = sorted(glob.glob('/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/csv_data
_2009_2019/*.csv'))
first = True
score_table = []
pred_table = []
i=1
for stock in files:
```

```
      # load the new file
      train_x_scaled, train_y_scaled, test_x_scaled,
test_y_scaled,train_x_scaler,train_y_scaler,test_x_scaler, test_y_scaler =
load_data(stock,days_to_predict)

      # design neural network
      model = Sequential()
      model.add(LSTM(200, input_shape=(train_x_scaled.shape[1],
train_x_scaled.shape[2])))
      model.add(Dropout(0.2))
    model.add(Dense(20,activation='relu'))
      model.add(Dense(1))

      logname = ("/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/training
_log/" + 'stock' +"%d"%i+'.log')
      i = i+1
      csv_logger = CSVLogger(logname)
    model.compile(loss='mse', optimizer='adam', metrics=['mse'])
      history = model.fit(train_x_scaled, train_y_scaled, epochs=200,
batch_size=16, validation_data=(test_x_scaled, test_y_scaled), verbose=0,
callbacks=[csv_logger],shuffle=False)


      pred = model.predict(test_x_scaled)
      test_scaled_back = test_y_scaler.inverse_transform(test_y_scaled)
      pred_sclade_back = test_y_scaler.inverse_transform(pred)

      #pred = model.predict(test_X)
      plt.plot( test_scaled_back, label="actual")
      plt.plot( pred_sclade_back, label="predictions ")
      plt.legend( loc='upper center', bbox_to_anchor=(0.5, -
0.05),fancybox=True, shadow=True, ncol=2)
      plt.title("30 days prediction using more complicated network")
      pyplot.show()

    pred_table.append(pred_sclade_back.tolist())
    print("done %s" %stock)


#ran the above functions separately in four times so have 4 tables
pred_table_df1 = pd.DataFrame(pred_table)
pred_table_df1.to_csv("/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/model2_p
red_table_1.csv", encoding='utf-8')

pred_table2 = []
i=73
for stock in files[73:]:

      # load the new file
      train_x_scaled, train_y_scaled, test_x_scaled,
test_y_scaled,train_x_scaler,train_y_scaler,test_x_scaler, test_y_scaler =
load_data(stock,days_to_predict)

      # design neural network
      model = Sequential()
```

```
      model.add(LSTM(50, input_shape=(train_x_scaled.shape[1],
train_x_scaled.shape[2])))
      model.add(Dropout(0.2))
      model.add(Dense(20,activation='relu'))
      model.add(Dense(1))

      logname = ("/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/training
_log/" + 'stock' +"%d"%i+'.log')
      i = i+1
      csv_logger = CSVLogger(logname)
      model.compile(loss='mse', optimizer='adam', metrics=['mse'])
      history = model.fit(train_x_scaled, train_y_scaled, epochs=50,
batch_size=16, validation_data=(test_x_scaled, test_y_scaled), verbose=0,
callbacks=[csv_logger],shuffle=False)


      pred = model.predict(test_x_scaled)
      test_scaled_back = test_y_scaler.inverse_transform(test_y_scaled)
      pred_sclade_back = test_y_scaler.inverse_transform(pred)

      #pred = model.predict(test_X)
      plt.plot( test_scaled_back, label="actual")
      plt.plot( pred_sclade_back, label="predictions ")
      plt.legend( loc='upper center', bbox_to_anchor=(0.5, -
0.05),fancybox=True, shadow=True, ncol=2)
      plt.title("30 days prediction using more complicated network")
      pyplot.show()

    pred_table2.append(pred_sclade_back.tolist())
      print(i)
      print("done %s" %stock)



pred_table_df2 = pd.DataFrame(pred_table2)
pred_table_df2.to_csv("/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/model2_p
red_table_2.csv", encoding='utf-8')

pred_table3 = []
i=352
for stock in files[352:]:

      # load the new file
      train_x_scaled, train_y_scaled, test_x_scaled,
test_y_scaled,train_x_scaler,train_y_scaler,test_x_scaler, test_y_scaler =
load_data(stock,days_to_predict)

      # design neural network
      model = Sequential()
      model.add(LSTM(50, input_shape=(train_x_scaled.shape[1],
train_x_scaled.shape[2])))
      model.add(Dropout(0.2))
      model.add(Dense(20,activation='relu'))
      model.add(Dense(1))
```

```
      logname = ("/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/training
_log/" + 'stock' +"%d"%i+'.log')
      i = i+1
      csv_logger = CSVLogger(logname)
      model.compile(loss='mse', optimizer='adam', metrics=['mse'])
      history = model.fit(train_x_scaled, train_y_scaled, epochs=50,
batch_size=16, validation_data=(test_x_scaled, test_y_scaled), verbose=0,
callbacks=[csv_logger],shuffle=False)


      pred = model.predict(test_x_scaled)
      test_scaled_back = test_y_scaler.inverse_transform(test_y_scaled)
      pred_sclade_back = test_y_scaler.inverse_transform(pred)

      #pred = model.predict(test_X)
      plt.plot( test_scaled_back, label="actual")
      plt.plot( pred_sclade_back, label="predictions ")
      plt.legend( loc='upper center', bbox_to_anchor=(0.5, -
0.05),fancybox=True, shadow=True, ncol=2)
      plt.title("30 days prediction using more complicated network")
      pyplot.show()

      pred_table3.append(pred_sclade_back.tolist())
      print(i)
  print("done %s" %stock)

pred_table_df3 = pd.DataFrame(pred_table3)
print(pred_table_df3.shape)
pred_table_df3.to_csv("/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/model2_p
red_table_3.csv", encoding='utf-8')

pred_table4 = []
test_loss = []
# for i in range(days_to_predict):
#     test_loss.append(0)
i=359

for stock in files[359:]:

      # load the new file
      train_x_scaled, train_y_scaled, test_x_scaled,
test_y_scaled,train_x_scaler,train_y_scaler,test_x_scaler, test_y_scaler =
load_data(stock,days_to_predict)

      # design neural network
      model = Sequential()
      model.add(LSTM(50, input_shape=(train_x_scaled.shape[1],
train_x_scaled.shape[2])))
      model.add(Dropout(0.2))
      model.add(Dense(20,activation='relu'))
      model.add(Dense(1))

      logname = ("/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/training
_log/" + 'stock' +"%d"%i+'.log')
```

```
      i = i+1
      csv_logger = CSVLogger(logname)
      model.compile(loss='mse', optimizer='adam', metrics=['mse'])
      history = model.fit(train_x_scaled, train_y_scaled, epochs=50,
batch_size=16, validation_split=0.1, verbose=0,
callbacks=[csv_logger],shuffle=False)

      test_pred_loss = model.evaluate(test_x_scaled, test_y_scaled,
batch_size =16)[0]
      test_loss.append(test_pred_loss)
      print(test_pred_loss)

      pred = model.predict(test_x_scaled)
      test_scaled_back = test_y_scaler.inverse_transform(test_y_scaled)
      pred_sclade_back = test_y_scaler.inverse_transform(pred)

      #pred = model.predict(test_X)
      plt.plot( test_scaled_back, label="actual")
      plt.plot( pred_sclade_back, label="predictions ")
      plt.legend( loc='upper center', bbox_to_anchor=(0.5, -
0.05),fancybox=True, shadow=True, ncol=2)
      plt.title("30 days prediction using more complicated network")
      pyplot.show()

    pred_table4.append(pred_sclade_back.tolist())
      print(i)

      print("done %s" %stock)

pred_table_df4 = pd.DataFrame(pred_table4)
print(pred_table_df4.shape)
pred_table_df4.to_csv("/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/model2_p
red_table_4.csv", encoding='utf-8')



#getting returns
last_price = []
for stock in files:
      last_price.append(read_csv(stock, header=0, index_col=0).iloc[-1,0])
last_price = np.array(last_price)

# Saving all prediction results
pre1 = read_csv('/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/preds/mo
del2_pred_table_1.csv')
pre2 = read_csv('/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/preds/mo
del2_pred_table_2.csv')
pre3 =read_csv('/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/model2_p
red_table_3.csv')
pre4 = read_csv('/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/model2_p
red_table_4.csv')
pre1.drop(pre1.columns[0], axis=1,inplace=True)
```

```
pre2.drop(pre2.columns[0], axis=1,inplace=True)
pre3.drop(pre3.columns[0], axis=1,inplace=True)
pre4.drop(pre4.columns[0], axis=1,inplace=True)

pred_all = pd.concat([pre1,pre2,pre3,pre4],ignore_index=True)
#pred_all.to_csv("/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/preds/pr
ed_table.csv", encoding='utf-8')

pred_all_with_last_price = pd.concat([pre1,pre2,pre3,pre4],ignore_index=True)
pred_all_with_last_price['-1'] = last_price
cols = pred_all_with_last_price.columns.tolist()
cols = cols[-1:]+cols[:-1]
pred_all_with_last_price = pred_all_with_last_price[cols]
pred_all_with_last_price.drop(pred_all_with_last_price.columns[30], axis=1,
inplace=True)
pred_all_with_last_price.head()
#Calculating Daily Net Returns using Predicted Prices and Descriptive Report
for Predictions for the next 30 days
#calculating daily net return
net_return = np.zeros((505,30))
net_return = (pred_all.values)/pred_all_with_last_price.values-1
#values
return_table = pd.DataFrame(net_return)
(return_table.describe())
#return_table.iloc[:,1:].mean(axis=0)
#return_table.to_csv("/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/preds/re
turn_table.csv", encoding='utf-8')
sp500 = read_csv("sp500.csv")
plt.plot(sp500.iloc[1:,-1], label='sp500 Index Actual Total Daily Net
Return')
plt.plot(return_table.iloc[:,1:].mean(axis = 0),label='Daily Average Net
Return')
plt.title("Daily Average Net Return of S&P500 stocks for the next 30 days")
plt.legend( loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True,
shadow=True, ncol=2)
plt.show

#####################################
#model2_ Multi_step_forecasting (using_predictions_to_predict), encoder-
decoder model
#####################################
# Importing the libraries
import os
from math import sqrt
import math
from numpy import array
import numpy as np
import pandas as pd
import datetime
import glob
import random
import datetime as dt
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
```

```
from numpy import newaxis
from numpy import split
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

from matplotlib import pyplot
import matplotlib.pyplot as plt

from keras import optimizers
from keras.layers import Dense, Activation, Dropout, LSTM, Embedding,
Flatten, RepeatVector, TimeDistributed
from keras.models import Sequential, load_model, model_from_json
from keras.callbacks import EarlyStopping, ModelCheckpoint

def to_supervised(train, n_input, n_out):
      # flatten data
      data = train.reshape((train.shape[0]*train.shape[1], train.shape[2]))
      X, y = list(), list()
      if(n_input > train.shape[0]*train.shape[1]):
      print("not enough data!")
      # step over the entire data history one time step at a time
      for in_start in range(len(data)):
      # define the end of the input sequence
      in_end = in_start + n_input
      out_end = in_end + n_out
      # ensure we have enough data for this instance
      if out_end < len(data):
            #include all of the features to train the model
            X.append(data[in_start:in_end, :])
            #recording target: the first column
              y.append(data[in_end:out_end, 0])
            #print(data[in_end:out_end, 0])

      train_x_scaler = MinMaxScaler()
      train_y_scaler = MinMaxScaler()

      x_train = np.array(X)
      #print(x_train.shape)
      train_s1,train_s2,train_s3 =
x_train.shape[0],x_train.shape[1],x_train.shape[2]
      x_train_scaled =
train_x_scaler.fit_transform(x_train.reshape(train_s1*train_s2,train_s3))
      x_train_scaled = x_train_scaled.reshape(train_s1,train_s2,train_s3)
      y_train = np.array(y)
      y_train_scaled = train_y_scaler.fit_transform(y_train)
      print("done to supervised")

  return x_train_scaled, y_train_scaled, train_x_scaler,train_y_scaler

  def fill_missing(values):
      #has to be converted to numeric values first
      day = 1
      for row in range(values.shape[0]):
```

```
            for col in range(values.shape[1]):
                  if np.isnan(values.iloc[row, col]):
                  if(np.isnan(values.iloc[row - day, col])):
                          values.iloc[row, col] = 0
                  else:
                        values.iloc[row, col] = values.iloc[row - day, col]

      return(values)


  def split_dataset(data,n_out,split=False, split_percent=None, number=False,
number_of_test=None):
      if split:
      # split into standard months
      row = int((len(data)*split_percent))
      train_size = row - row%(n_out)
      test_size = (len(data) - train_size) - ((len(data) -
train_size)%(n_out))
      train, test = data[0:train_size],
data[train_size:(train_size+test_size)]
      if number:
      train_size = (len(data) - number_of_test) - (len(data) -
number_of_test)%n_out
      test_size = number_of_test - number_of_test%n_out
      train, test = data[0:train_size],
data[train_size:(train_size+test_size)]

      # restructure into windows of daily data
      train = array(np.split(train, len(train)/n_out)) #how many as to output
      test = array(np.split(test, len(test)/n_out))
      return train, test, train_size, test_size



  # evaluate one or more weekly forecasts against expected values
  def evaluate_forecasts(actual, predicted,error_type):

      scores = list()
      # calculate an RMSE score for each day
      for i in range(actual.shape[1]):
      #calculate mae
      if error_type == 'mae':
            error = mean_absolute_error(actual[:, i], predicted[:, i])
      # calculate rmse
      elif error_type == 'rmse':
            # calculate mse
            mse = mean_squared_error(actual[:, i], predicted[:, i])
            error = sqrt(mse)
            #print(rmse)
            # store
            #scores.append(rmse)
      scores.append(error)
      # calculate overall RMSE
      s = 0
      for row in range(actual.shape[0]):
      for col in range(actual.shape[1]):
            s += (actual[row, col] - predicted[row, col])**2
```

```
      score = sqrt(s / (actual.shape[0] * actual.shape[1]))
      print("evaluating forecasts")
      return score, scores
  # train the model
  def build_model(train_x, train_y, n_input,n_out,params):
      # get parameters from params
      verbose, epochs, batch_size = 1, params["epochs"], params["batch_size"]
      n_timesteps, n_features, n_outputs = train_x.shape[1],
train_x.shape[2], train_y.shape[1]
      optimizer = params["optimizer"]
      loss = params["loss"]

      # reshape output into [samples, timesteps, features]
      train_y = train_y.reshape((train_y.shape[0], train_y.shape[1], 1))
      # define model
      model = Sequential()
      print("building LSTM model")
      #define an LSTM hidden layer with 100 units; input_shape=(n_timesteps,
n_features)
      model.add(LSTM(100,
activation='relu',input_shape=(train_x.shape[1],train_x.shape[2]))) #the
encoder to read and encode the input sequence
      #presented to the LSTM decoder

      #the RepeatVector is used as an adapter to fit the fxed-sized 2D output
of the encoder to the differing length and 3D input expected by the decoder
      #https://keras.io/layers/core/#repeatvector
      model.add(RepeatVector(n_outputs))
      model.add(Dropout(0.2))
      #model.add(LSTM(60, dropout=0.0))
      #model.add(Dropout(0.4))

      #early stopping
      #es = EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=40, min_delta=0.0001)

      #then define the decoder as an LSTM hidden layer with 200 units
      #each of the 50 units will output a value for each of the 7 days,
      #representing the basis for what to predict for each day in the output
sequence.
      model.add(LSTM(50,activation='relu',return_sequences=True)) # decoder
that will read the encoded input sequence and make a one-step prediction for
each element in the output sequence
      model.add(Dropout(0.2))
      #The TimeDistributed wrapper allows the same output layer to be reused
for each element in the output sequence.
      model.add(TimeDistributed(Dense(100, activation='tanh')))#,
activation='relu'
      model.add(TimeDistributed(Dense(1)))
      #optimizer = optimizers.RMSprop(lr=0.00010000)

      model.compile(loss=loss, optimizer=optimizer) #mean_squared_error
      #A metric function is similar to a loss function, except that the
results from evaluating a metric are not used when training the model.
      #You may use any of the loss functions as a metric function.
      # fit network
     print("fitting LSTM model")
```

```
      history = model.fit(train_x, train_y,validation_split=0.1,
epochs=epochs, batch_size=batch_size, verbose=verbose) #add validation data?

      return model,history


  # make a forecast
  def forecast(model, history, n_input):
      # flatten data
      data = np.array(history)
      data = data.reshape((data.shape[0]*data.shape[1], data.shape[2]))
      # retrieve last observations for input data
      input_x = data[-n_input:, :]
      # reshape into [1, n_input, n]
      input_x = input_x.reshape((1, input_x.shape[0], input_x.shape[1]))
      # forecast the n_out days
      yhat = model.predict(input_x, verbose=0)
      #print(yhat.shape)
      # we want the forecast in vector form
      y = yhat
      yhat = yhat[0]

      #print(yhat.shape)
      #print("forecasting")
      return yhat,y

  # evaluate a single model
  def evaluate_model(train, test, n_input):
      # fit model
      model = build_model(train, n_input)
      # history is a list of weekly data
      history = [x for x in train]
      # walk-forward validation over each week
      predictions = list()
      for i in range(len(test)):
      # predict the week
      yhat_sequence = forecast(model, history, n_input)
      # store the predictions
      predictions.append(yhat_sequence)
      # get real observation and add to history for predicting the next week
      history.append(test[i, :])
      # evaluate predictions days for each week
      predictions = array(predictions)
      score, scores = evaluate_forecasts(test[:, :, 0], predictions,'rmse')
      print("evaluting model")
      return score, scores


  #Loading Data and Normalizing¶
  cols = [
   'PX_CLOSE_1D',
      'DAY_TO_DAY_TOT_RETURN_NET_DVDS',
   'CUR_MKT_CAP',
   'LAST_CLOSE_TRR_6MO',
   'VOLATILITY_180D',
   'SHORT_INT_RATIO',
   'NUM_INSIDERS_SELLING_SHARES',
```

```
   'NUM_INSIDERS_BUYING_SHARES',
   'NUM_INSIDERS_OWNING_SHARES',
   'PCT_INSIDER_SHARES_OUT',
   'PX_TO_CASH_FLOW',
   'DIVIDEND_12_MONTH_YIELD',
   'PE_RATIO',
   'CURR_ENTP_VAL',
   'CF_FREE_CASH_FLOW',
   'PX_TO_FREE_CASH_FLOW',
   'PX_TO_BOOK_RATIO',
   'IS_OPERATING_MARGIN',
   'PROF_MARGIN',
   'GROSS_MARGIN',
   'SALES_REV_TURN',
   'NET_REV',
   'RETURN_ON_ASSET',
   'RETURN_COM_EQY',
   'GEO_GROW_ROE',
   'GEO_GROW_NET_SALES',
   'EPS_GROWTH',
   'REVENUE_GROWTH_ADJUSTED_YOY',
   'DILUTED_EPS_5YR_AVG_GR',
   'EQY_DPS_NET_5YR_GROWTH',
   'DEBT_TO_MKT_CAP']
  dataset = read_csv('/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/csv_data
_2009_2019/AAPL UW Equity.csv', header=0, infer_datetime_format=True,
parse_dates=['Dates'], index_col=0)
  dataset = dataset.loc[:, cols]
  dataset = dataset.ffill(axis=0)
  dataset = dataset.bfill(axis=0)
  dataset = dataset.fillna(dataset.mean())
  dataset = dataset.fillna(0)
  dataset = dataset.astype('float32')
  print(dataset.describe())

  n_out = 10
  n_input = 100 # use n_input inputs to predict n_out ahead
  train, test,train_size, test_size= split_dataset(dataset.values,
n_out,split= True,  split_percent = 0.9, number = False, number_of_test = 30)
  #train, test,train_size, test_size= split_dataset(dataset.values,
n_out,split = False, split_percent = 0.95, number = True, number_of_test =
30)

  scaler = MinMaxScaler(feature_range=(-1, 1))
  scaled = scaler.fit_transform(dataset.values)
  #train, test = split_dataset(dataset.values, n_out, 0.8) #~8 years to train

  train_s1, train_s2,train_s3 = train.shape[0], train.shape[1],train.shape[2]
  test_s1,test_s2,test_s3 = test.shape[0], test.shape[1],test.shape[2]

  t_scaler = MinMaxScaler(feature_range=(0, 1))
  train_scaler = MinMaxScaler()
  test_scaler = MinMaxScaler()

  #scaled train data
```

```
  data_train =
t_scaler.fit_transform(train.reshape(train_s1*train_s2,train_s3))
  data_train = data_train.reshape(train_s1,train_s2,train_s3)
  #scaled test data
  data_test = t_scaler.fit_transform(test.reshape(test_s1*test_s2,test_s3))
  data_test = data_test.reshape(test_s1,test_s2,test_s3)


  x_train, y_train, train_x_scaler, train_y_scaler = to_supervised(train,
n_input,n_out)
  x_test, y_test, test_x_scaler, test_y_scaler =
to_supervised(test,n_input,n_out)

  print("Are any NaNs present in train/test matrices?",np.isnan(train).any(),
np.isnan(test).any())
  print(np.max(y_train))
  print("x_train:",x_train.shape,"y_train:", y_train.shape,"x_test:",
x_test.shape, "y_test:",y_test.shape, "data_train:", data_train.shape,
"data_test:",  data_test.shape )

  #building model
  params = {
       "batch_size": 16,
       "epochs":30,
       "optimizer": 'adam',
       "loss": 'mse'
  }
```

#how many as a group ; number of prior observations that the model will use as input in order to make a prediction.

```
  model,history = build_model(x_train, y_train, n_input, n_out,params)

  # save model to single file
  model.save('lstm_model.h5')

  #model = load_model('lstm_model.h5')
  print(model.summary())


  #plotting results
  #print(history.history.keys()) #check for available keys
  plt.figure()
  plt.plot(history.history['loss'])
  plt.plot(history.history['val_loss'])
  plt.title('Model loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Test'], loc='upper left')
  print("training loss:",history.history['loss'])
  print("validation loss:",history.history['val_loss'])

  #calculating evaluation metrics
  history_train = [x for x in x_train]
  # walk-forward validation over each day
  predictions = list()
  predictions_full = list()
```

```
  for i in range(len(x_test)):
       # predict the week
       yhat_sequence,y = forecast(model, history_train, n_input)

       # store the predictions
       predictions_full.append(y)
       predictions.append(yhat_sequence)

       # get real observation and add to history for predicting the next
day/days
       history_train.append(x_test[i, :])

  # evaluate predictions days for each day/period
  predictions = array(predictions)
  predictions_full = array(predictions_full)
  score, scores = evaluate_forecasts(y_test[:, :], predictions,'mae')
  #plt.subplot(211)
  plt.title('Model RMSE for %d Days' %n_out)
  plt.plot(scores, marker='o', label='lstm')
  plt.show()
  print("RMSE for each pred is:",scores)

#ploting scaled-back predictions
  plt.plot(dataset.iloc[:,0])

  #ploting predictions
  y_pred = (model.predict(x_test))
  y_pred = y_pred.reshape(y_pred.shape[0],y_pred.shape[1])
  pred_scaled_back = test_y_scaler.inverse_transform(y_pred)

  #test_loss = model.evaluate(x_test, y_test, batch_size =16)

  actual_scaled_back = test_y_scaler.inverse_transform(y_test)
  plt.plot(actual_scaled_back[-1,:],label="actual")
  plt.plot( pred_scaled_back[-1,:], label="predictions ")
  #plt.title("Using Predictions to Predict ahead: %d days ahead of AAPL
Stcok" % d)
  plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True,
shadow=True, ncol=2)
  pyplot.show()
  p = (model.predict(x_test))
  print(p.shape)
  print(x_test.shape)
  print(y_test.shape)

  #plotting errors & other metrics
  #x_test.shape
  y_test_shape = y_test.reshape(y_test.shape[0],y_test.shape[1],1)
  test_loss = model.evaluate(x_test, y_test_shape, batch_size =16)
  print('Avaerage training MSE is:', np.mean(history.history['loss']))

  print("Average validation MSE is:", np.mean(history.history['val_loss']))

  print("Test MSE is:",test_loss)
  #Getting Prediction for the future
  pred_table = []
  actual_table = []
```

```
  for i in range(3,0,-1):
      #print(i)
      if i==1:
      data_actual = (dataset.iloc[(-n_out)*i:,:]).values
      data_for_future = t_scaler.fit_transform((dataset.iloc[(-
n_input)*i:,:]).values)
      data_for_future = data_for_future.reshape((1,n_input,31))
      end = None
      else:
      data_actual = (dataset.iloc[(-n_out)*i:(-n_out*(i-1)),:]).values
      data_for_future = t_scaler.fit_transform((dataset.iloc[(-n_input)*i:(-
n_input*(i-1)),:]).values)
      data_for_future = data_for_future.reshape((1,n_input,31))
      end = (-n_out*(i-1))

      pred_next = model.predict(data_for_future)[0]
      rowLen = pred_next.shape[0]
      matrix = np.zeros((rowLen,(dataset.shape[1]-1)))
      pred_matrix =  np.concatenate((pred_next,matrix),axis=1)
      pred_next = t_scaler.inverse_transform(pred_matrix)[:,0]
      pred_table.append(pred_next)
      data_actual_y = (dataset.iloc[(-n_out)*i:end,0]).values
      actual_table.append(data_actual_y)

  pred_table_flatten = np.concatenate(pred_table).ravel()
  actual_table_flatten = np.concatenate(actual_table).ravel()

  plt.plot(actual_table_flatten,label="actual")
  plt.plot(pred_table_flatten, label="predictions")
  plt.title("Scaled back prices of %s : %d days prediction using
model2" %("AAPL:", n_out*3))
  plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True,
shadow=True, ncol=2)
  plt.show()

  #Parameter Selection
  #1.Hand tuning
  #2.Grid Searching
  #3.Random Search—This a subset of Grid Search which randomly selects subset
of all possible combinations.
  #4.Bayesian Optimization/Other probabilistic optimizations

#####################################
#model3_LSTM_direct_predictions_different_parameters_each_pred
Selected the number of steps to predict ahead = 3 and built 3 LSTM models.
For each model used different variable (fit0, fit1, fit2) to avoid any
“memory leakage” between models.
The model initialization code is the same for all 3 models except changing
parameters (number of neurons in LSTM layer)
#####################################
  # Importing the libraries
  import os
  import math
  from numpy import array
  import numpy as np
  import pandas as pd
  import datetime
```

```
  import glob
  import random
  import datetime as dt
  from pandas import read_csv
  from pandas import DataFrame
  from pandas import concat
  from numpy import newaxis
  from numpy import split
  from sklearn import preprocessing
  from keras.layers import Dense, Activation, Dropout, LSTM, Embedding,
Flatten, RepeatVector, TimeDistributed
  from keras.models import Sequential, load_model, model_from_json
  from keras.callbacks import EarlyStopping, ModelCheckpoint
  from sklearn.preprocessing import MinMaxScaler
  from sklearn.preprocessing import LabelEncoder
  from matplotlib import pyplot
  import matplotlib.pyplot as plt
  from sklearn.metrics import mean_squared_error

  cols = ['PX_CLOSE_1D',
       'DAY_TO_DAY_TOT_RETURN_NET_DVDS',
   'CUR_MKT_CAP',
   'LAST_CLOSE_TRR_6MO',
   'VOLATILITY_180D',
   'SHORT_INT_RATIO',
   'NUM_INSIDERS_SELLING_SHARES',
   'NUM_INSIDERS_BUYING_SHARES',
   'NUM_INSIDERS_OWNING_SHARES',
   'PCT_INSIDER_SHARES_OUT',
   'PX_TO_CASH_FLOW',
   'DIVIDEND_12_MONTH_YIELD',
   'PE_RATIO',
   'CURR_ENTP_VAL',
   'CF_FREE_CASH_FLOW',
   'PX_TO_FREE_CASH_FLOW',
   'PX_TO_BOOK_RATIO',
   'IS_OPERATING_MARGIN',
   'PROF_MARGIN',
   'GROSS_MARGIN',
   'SALES_REV_TURN',
   'NET_REV',
   'RETURN_ON_ASSET',
   'RETURN_COM_EQY',
   'GEO_GROW_ROE',
   'GEO_GROW_NET_SALES',
   'EPS_GROWTH',
   'REVENUE_GROWTH_ADJUSTED_YOY',
   'DILUTED_EPS_5YR_AVG_GR',
   'EQY_DPS_NET_5YR_GROWTH',
   'DEBT_TO_MKT_CAP']
  steps_to_predict = 30
  train_loss=[]
  test_loss=[]
  forecast=[]
  hist = []
  for i in range(steps_to_predict):
      train_loss.append(0)
```

```
      test_loss.append(0)
      forecast.append(0)

dataset = read_csv('/Users/yangt/OneDrive - Cornell
University/Spring/Project/past/confidence.based.security.forecasting/csv_data
_2009_2019/AAPL UW Equity.csv', header=0, infer_datetime_format=True,
parse_dates=['Dates'], index_col=0)
  #dataset = fill_missing(dataset)
  dataset = dataset.loc[:, cols]
  dataset = dataset.ffill(axis=0)
  dataset = dataset.fillna(dataset.mean())
  dataset = dataset.fillna(0)
  dataset = dataset.astype('float32')
  #print(dataset.describe())
  dataset = dataset.ffill(axis=0)
  dataset = dataset.fillna(dataset.mean())
  dataset = dataset.fillna(0)


  print("Are any NaNs present in train/test
matrices?",np.isnan(dataset.values).any())

  data_to_use= len(dataset)-10
  total_data = len(dataset)
  # number of training data
  start = 0
  end =  data_to_use - steps_to_predict
  train_end = len(dataset)-steps_to_predict-1
  dataset = dataset[cols]
  y_original = dataset.iloc[:,0]
  x_variables = dataset.iloc[:,1:]
  y_totalReturn = dataset.iloc [start:total_data ,0]    #daily return

  for i in range(steps_to_predict):
      if i==0:
      units=50
      batch_size= 16
      epoch_size = 50
      if i==1:
      units=20
      batch_size= 32
      epoch_size = 25
      if i==2:
      units=100
      batch_size= 16
      epoch_size = 25
      if i==59:
      units=10
      batch_size= 16
      epoch_size = 20
      else:
      units=10
      batch_size= 48
      epoch_size = 10

      yt_ = y_totalReturn.shift (- i - 1)
      data = pd.concat([ yt_, y_totalReturn,  x_variables], axis =1)
```

```
      data.columns.values[0] ='Return_Target'
      data.dropna(inplace=True)
      cols = data.columns[1:]
      #x = data.loc[:,cols]
      x = data[cols]
      y = data['Return_Target']
      #yt = data.iloc[:,0]

      scaler_x = preprocessing.MinMaxScaler ()
      x = np.array (x).reshape ((len( x) ,len(cols)))
      x = scaler_x.fit_transform (x)

      scaler_y = preprocessing. MinMaxScaler ( )#feature_range =( -1, 1)
      y = np.array (y).reshape ((len( y), 1))
     y = scaler_y.fit_transform (y)


      x_train = x[0: train_end,]
      x_test = x[(train_end ):len(x),]
      y_train = y[0: train_end]
      y_test = y[(train_end):len(y)]

     print("x_train shape:", x_train.shape, "x_test shape:", x_test.shape,
"y_train shape:", y_train.shape,"y_test shape:", y_test.shape)

      #seed =2019
      #np.random.seed (seed)

    #model initialization
      if (i == 0) :
      prediction_data=[]
      for j in range (len(y_test)) :
                  prediction_data.append(0)
      #converting to 3D
      x_train = x_train.reshape(x_train.shape + (1,))
      x_test = x_test.reshape(x_test.shape + (1,))
      ###############
      ##  i=0     ##
      ###############
      if (i == 0):
      fit0 = Sequential()
      fit0.add (LSTM (units , activation = 'tanh', inner_activation =
'hard_sigmoid' , input_shape =(x_train.shape[1], x_train.shape[2]) ))
      fit0.add(Dropout(0.2))
      fit0.add (Dense (1))
      fit0.compile(loss ="mse" , optimizer = "adam")
      history1 = fit0.fit(x_train, y_train, batch_size =batch_size, nb_epoch
=epoch_size, shuffle = False,validation_split=0.1)
      hist.append(history1.history)
      train_loss[i] = fit0.evaluate(x_train, y_train, batch_size =batch_size)
      test_loss[i] = fit0.evaluate(x_test, y_test, batch_size =batch_size)

      pred = fit0.predict(x_test)
      pred =
scaler_y.inverse_transform(np.array(pred).reshape((len(pred),1)))
```

```
      # below is just for i == 0, not predicting steps ahead

      for j in range(len(pred) - 0 ) :
            prediction_data[j] = pred[j]

      forecast[i]=pred[-1]

      ###############
      ##  i=1     ##
      ###############
      if i == 1 :
      fit1 = Sequential ()
      fit1.add(LSTM (units , activation = 'relu' , input_shape
=(x_train.shape[1], x_train.shape[2])))
      fit1.add(Dropout(0.2))
      fit1.add(Dense (1))
      fit1.compile(loss ="mse" , optimizer = "adam")
      history2 = fit1.fit(x_train, y_train, batch_size =batch_size,
nb_epoch=epoch_size, shuffle = False,validation_split=0.1)
      hist.append(history2.history)
      train_loss[i] = fit1.evaluate(x_train, y_train, batch_size =batch_size)
      test_loss[i] = fit1.evaluate(x_test, y_test, batch_size =batch_size)

      pred = fit1.predict (x_test)
      pred = scaler_y.inverse_transform (np. array (pred). reshape
((len( pred), 1)))
      forecast[i]=pred[-1]


      ###############
      ##  i=2     ##
      ###############
      if i==2 :
      fit2 = Sequential()
      fit2.add (LSTM (units , activation = 'relu', input_shape
=(x_train.shape[1], x_train.shape[2])))
      fit2.add(Dropout(0.2))
      fit2.add (Dense (1))
      fit2.compile(loss ="mse" , optimizer = "adam")
      history3 = fit2.fit(x_train, y_train, batch_size =batch_size, nb_epoch
=epoch_size, shuffle = False,validation_split=0.1)
      hist.append(history3.history)
      train_loss[i] = fit2.evaluate(x_train, y_train, batch_size=batch_size)
      test_loss[i] = fit2.evaluate(x_test, y_test, batch_size =batch_size)
      pred = fit2.predict(x_test)
      pred = scaler_y.inverse_transform(np.array(pred).reshape((len(pred),
1)))

      forecast[i]=pred[-1]

      else:

      fit = Sequential()
      fit.add (LSTM (units , activation = 'relu' , input_shape
=(x_train.shape[1], x_train.shape[2])))
      fit.add(Dropout(0.2))
      fit.add(Dense(1))
```

```
      fit.compile(loss ="mse" , optimizer = "adam")
      history4 = fit.fit(x_train, y_train, batch_size =batch_size, nb_epoch
=epoch_size, shuffle = False,validation_split=0.1)
      hist.append(history4.history)
      train_loss[i] = fit.evaluate(x_train, y_train, batch_size=batch_size)
      test_loss[i] = fit.evaluate(x_test, y_test, batch_size =batch_size)
      pred = fit.predict(x_test)
      pred = scaler_y.inverse_transform(np.array(pred).reshape((len(pred),
1)))

      forecast[i]=pred[-1]

      x_test = scaler_x.inverse_transform
(np.array(x_test).reshape((len(x_test), len(cols))))

  prediction_data_array = np.asarray(prediction_data)

  prediction_data_array = prediction_data_array.ravel()

  for j in range (len(prediction_data_array) - 1 ):
      prediction_data_array[len(prediction_data_array) - j - 1  ] =
prediction_data_array[len(prediction_data_array) - 1 - j - 1]

  prediction_data_forecast_combined = np.append(prediction_data_array,
forecast)
  x_test_all = y_original[len(y_original)-
len(prediction_data_forecast_combined)-1:len(y_original)-1]
  x_test_all = x_test_all.ravel()
  print(x_test_all.shape)
  plt.plot(x_test_all, label="actual")
  plt.plot(prediction_data_forecast_combined, label="predictions")


  plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True,
shadow=True, ncol=2)

  import matplotlib.ticker as mtick
  fmt = '$%.0f'
  tick = mtick.FormatStrFormatter(fmt)
  ax = plt.axes()
  ax.yaxis.set_major_formatter(tick)
  plt.show()

  #plotting forecasts
  plt.plot(x_test_all[-30:], label="actual")
  plt.plot(forecast, label="predictions")
  plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.05),fancybox=True,
shadow=True, ncol=2)
  plt.title("AAPL: Using Different Parameters for Predicting 30 Days Ahead")
  plt.show()


  val_loss=[]
  for history in hist:
      val_loss.append(history['val_loss'])
  loss = []
  for history in hist:
```

```
    loss.append(history['loss'])

print ("prediction data")
print ((prediction_data))

print ("x_test_all")
print ((x_test_all))

print ("avertage training mse")
print (np.mean(train_loss))
print ("average testing mse")
print (np.mean(test_loss))
print ("train_mae")
print (train_loss)
print ("test_mae")
print (test_loss)

# plot history
pyplot.plot(history1.history['loss'], label='train')
pyplot.plot(history1.history['val_loss'], label='test')
pyplot.title('model train vs validation loss')
pyplot.ylabel('loss')
pyplot.xlabel('epoch')
pyplot.legend()
pyplot.show()

hist_arr=(np.asarray(hist))
valloss = 0
trainloss = 0
for i in range(len(hist_arr)):
    valloss = valloss + np.mean(hist_arr[i]['val_loss'])
    trainloss = trainloss + np.mean(hist_arr[i]['loss'])
np.mean(test_loss)
print("Average validation loss is:",valloss/len(hist_arr))
print("Average training loss is: ",trainloss/len(hist_arr))
```