

Stock Prediction

Python Programming Final Project

N.J A

H.E L

J.Y P

Table of Contents

0. Abstract

1. Introduction

2. Development

i. Setting Environment

1 Library import

2 Dictionary definition

ii. Initializing class definition

iii. Dataset creation

iv. Model construction

v. Prediction

vi. Plotting

vii. Example

viii. Dashapp

3. Conclusion

i. Summary

ii. Analysis of result

4. Bibliography

[Abstract]

In this project, the stock price of the stock market for a specific company will be predicted and visualized. Stock price is being predicted by building a model using historical stock price data, and the flow of stock price will be displayed in plot. One of the Recurrent Neural Network (RNN) model, Long Short Term Memory (LSTM) neural network model is used for this process. Finally, for the Graphical User Interface (GUI), we create a website which shows the plot by using Dashapp.

1. Introduction

Stock market is one of the most fascinating inventions of our time. They have had a significant impact on many areas like business, education, jobs, technology and thus on the economy. Over the years, investors and researchers have been interested in developing and testing models of stock price behavior. However, analyzing stock market movements and price behaviors is extremely challenging because of the markets dynamic, nonlinear, nonstationary, nonparametric, noisy, and chaotic nature. Stock markets are also affected by many highly interrelated factors that include economic, political, psychological, and company-specific variables.¹

Models for example, artificial Neural Network (ANN), Deep Neural Network (DNN), Convolutional Neural Network (CNN), Long Short Term Memory (LSTM) are commonly being used for stock price prediction.

ANN (Artificial Neural Networks) is a mathematical model or computational model that is inspired by the structural and functional aspects of biological neural networks. A single neural network call, or a perceptron, has an interconnected group of artificial neurons, which process computational information by using a connectionist approach from node to node.

DNN (Deep Neural Network) is generally a stack of multiple hidden layers instead of only one hidden layer in the standard ANN architecture. The DNN hidden layers are the multiple feed-forward layers that are trained with a back-propagation stochastic gradient descent. DNN has features such as an adaptive learning rate, rate annealing, momentum training, drop out, and regularization. These features are believed to be able to have a higher predictive accuracy compared to the regular ANN.

¹ Shah, D.; Isah, H.; Zulkernine, F. Stock Market Analysis: A Review and Taxonomy of Prediction Techniques. *Int. J. Financial Stud.* **2019**, *7*, 26. <https://doi.org/10.3390/ijfs7020026>

CNN (Convolutional Neural Network) is basically composed of layers of convolutions consisting of neurons, with tanh, ReLU being applied to the results. CNN uses convolutions over the input layer to compute the output. An individual layer of CNN applies different types of filters. The edges of the layers capture the shape of the data, and then they use these shapes to determine higher-level features. The last layer classifies the output by using these high-level features.

RNN(The Recurrent Neural Network) makes use of sequential information, The RNN the inputs and outputs as a dependent variable based on a time sequence. RNN performs the same task for every element of a sequence. The output at the last time step of RNN is dependent on the previous computations. RNN may be considered to have a “memory”, as it can capture information about calculations in past sequences. However, RNN has a limitation in capturing the length of the data. This leads to the development of the LSTM network, which can capture longer sequences of information.²

In this project, we will try to predict the stock price of a specific company based on time series data of stock price from Yahoo Finance using Long Short Term Memory (LSTM) model from Recurrent Neural Network (RNN). LSTM model is generally thought to be useful to analyze and predict time series data.

Recurrent networks can in principle use their feedback connections to store representations of recent input events in the form of activations. This is significant for many applications. The most widely used algorithms for learning what to put in short-term memory, however, took too much time or did not work well at all, especially when minimal time lags between inputs and corresponding teacher signals are long. So, LSTM was a novel recurrent network architecture in conjunction with an appropriate gradient-based learning algorithm. LSTM is designed to overcome these error back-flow problems. It can learn to bridge time intervals in excess of 1000 steps even in case of noisy, incompressible input sequences, without loss of short time lag

² Khumprom, P.; Yodo, N. A Data-Driven Predictive Prognostic Model for Lithium-ion Batteries based on a Deep Learning Algorithm. *Energies* **2019**, *12*, 660. <https://doi.org/10.3390/en12040660>

capabilities. This is achieved by an efficient, gradient-based algorithm for an architecture enforcing constant error flow through internal states of special units.³

'Dashapp' library, which is used for GUI, is a framework for visualizing data. It can show the result of analysis in a website based on user actions. For this project, we display the graph of predicted stock prices using the LSTM stock prediction model.

The objective of this project is to create a neural network model capable of predicting the stock prices for the most important companies assets.

2. Development

i. Setting environment

1 Library import

Code :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
import plotly.graph_objects as go
from dash import dcc
from dash import html
from dash.dependencies import Input, Output
from jupyter_dash import JupyterDash
import dash_bootstrap_components as dbc
```

Import the libraries needed for implementation. Numpy, pandas, matplotlib, sklearn, tensorflow, plotly are being used for this project.

2 Dictionary definition

Code :

³ Hochreiter S , Schmidhuber J . "Long Short-Term Memory" Neural computation : 1735-1780.

```

companies = {
    "AAPL": "Apple",
    "MSFT": "Microsoft",
    "AMZN": "Amazon",
    "GOOG": "Alphabet",
    "FB": "Facebook",
    "BRK": "Berkshire Hathaway",
    "JNJ": "Johnson & Johnson",
    "V": "Visa",
    "JPM": "JPMorgan Chase",
    "PG": "Procter & Gamble",
    "NVDA": "NVIDIA",
    "MA": "Mastercard",
    "HD": "Home Depot",
    "UNH": "UnitedHealth Group",
    "DIS": "Walt Disney",
    "BAC": "Bank of America",
    "PYPL": "PayPal",
    "TSLA": "Tesla",
    "VZ": "Verizon",
    "NFLX": "Netflix",
    "ADBE": "Adobe",
    "CMCSA": "Comcast",
    "INTC": "Intel",
    "CSCO": "Cisco",
    "PFE": "Pfizer",
    "WMT": "Walmart",
    "ABT": "Abbott Laboratories",
    "ABBV": "AbbVie",
    "CRM": "Salesforce",
    "COST": "Costco Wholesale",
    "T": "AT&T"
}

```

Defining a dictionary which includes names of companies as value and key for each company.

ii. Initializing class definition

Code :

```

class Finpred:
    def __init__(self, company_name, end_date, num_prev_days):
        self.company_name = company_name
        self.end_date = end_date
        self.num_prev_days = num_prev_days

```

Class named 'Finpred' is starting to be built from this part. The '__init__' method initializes the

instance variables of the class. Variable named `company_name` is for the name of the company whose stock prices will be predicted. Variable `end_date` is the end date for downloading historical stock price data and `num_prev_days` is the number of previous days' stock prices to consider for prediction.

iii. Dataset creation

Code :

```
def df(self):  
    return  
pd.read_csv(f'https://query1.finance.yahoo.com/v7/finance/download/{self.company_name}?period1=0&period2={self.end_date}&interval=1d&events=history')  
  
def load_data(self):  
    # Process the data  
    df = self.df()  
    df['Date'] = pd.to_datetime(df['Date'])  
    df = df.set_index('Date')  
    df = df[['Close']]  
  
    # Normalize the data  
    scaler = MinMaxScaler(feature_range=(0, 1))  
    scaled_data = scaler.fit_transform(df.values)  
  
    # Split into train and test sets  
    train_size = int(len(scaled_data) * 0.8)  
    train_data = scaled_data[:train_size, :]  
    test_data = scaled_data[train_size-self.num_prev_days:, :]  
  
    # Prepare training and testing data  
    self.x_train, self.y_train = self.create_dataset(train_data)  
    self.x_test, self.y_test = self.create_dataset(test_data)  
  
def create_dataset(self, data):  
    x, y = [], []  
    for i in range(self.num_prev_days, len(data)):  
        x.append(data[i - self.num_prev_days:i, 0])  
        y.append(data[i, 0])  
    return np.array(x), np.array(y)
```

First, 'df' method retrieves historical stock price data from Yahoo Finance for the specified company using the pandas library. It constructs a URL with the `company_name` and `end_date`, and fetches the

data as a pandas data frame, and returns it.

For the 'load_data' method, this method processes the retrieved stock price data. It first calls the 'df' method to fetch the data. Second, it converts the 'Date' column to a 'datetime' data type and sets the 'Date' column as the index of the data frame. Then, selects only the 'Close' column, which represents the closing prices. Next, it normalizes the closing prices using the MinMaxScaler from sklearn.preprocessing. And the normalized data is split into train and test data sets, 80% for training and 20% for testing. Finally by calling the 'create_dataset' method, input-output pairs for training and testing are created.

Method 'create_dataset' takes in a numpy array of data and creates input-output pairs for the RNN model. It slices the data based on the number of previous days specified. Each input contains the closing prices of the preceding days, and the corresponding output is the closing price of the next day.

iv. Model construction

Code :

```
def build_model(self):
    # Build the RNN model
    self.model = Sequential()
    self.model.add(LSTM(units=50, return_sequences=True,
input_shape=(self.num_prev_days, 1)))
    self.model.add(LSTM(units=50))
    self.model.add(Dense(units=1))

    # Compile the model
    self.model.compile(optimizer='adam', loss='mean_squared_error')

    def train_model(self):
        # Train the model
        self.model.fit(self.x_train, self.y_train, epochs=5,
batch_size=32)
```

Method 'build_model' constructs the RNN model using the Keras library. It defines a sequential model with two LSTM layers followed by a dense layer. The model is compiled with the Adam optimizer and the means squared error (MSE) loss function.

The 'train_model' method trains the RNN model using the training data. It fits the model to the input-output pairs with 5 epochs and batch size 32.

v. Prediction

Code :

```
def predict(self):
    # Make predictions
    train_predictions = self.model.predict(self.x_train)
    test_predictions = self.model.predict(self.x_test)

    # Transform the predictions to original scale
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaler.fit(pd.DataFrame(self.df()['Close']))
    train_predictions = scaler.inverse_transform(train_predictions)
    test_predictions = scaler.inverse_transform(test_predictions)

    return train_predictions, test_predictions
```

This method makes predictions using the trained model. It uses the trained model to predict the closing prices for both the training and testing datasets. As the predictions were initially scaled, the method inversely transforms them to the original scale using the MinMaxScaler. Finally, it returns the predictions for both the training and testing datasets.

vi. Plotting

Code :

```
def plot_predictions(self, train_predictions,
test_predictions, company):
    df=self.df()
    train_actual =
df.iloc[self.num_prev_days:len(train_predictions) + self.num_prev_days,
:]['Close'].values
    test_actual = df.iloc[len(train_predictions) +
self.num_prev_days:, :]['Close'].values
    train_predictions = train_predictions.flatten()
    test_predictions = test_predictions.flatten()
    index=pd.date_range(df['Date'][0],
df['Date'][len(df['Date'])-1], freq='D')

    fig = go.Figure()
```

```

fig.add_trace(go.Scatter(x=index[self.num_prev_days:len(train_predictions) + self.num_prev_days], y=train_actual, name='Actual Price Train'))

fig.add_trace(go.Scatter(x=index[self.num_prev_days:len(train_predictions) + self.num_prev_days], y=train_predictions, name='Predicted Price Train'))

fig.add_trace(go.Scatter(x=index[len(train_predictions) + self.num_prev_days:], y=test_actual, name='Actual Price Test'))

fig.add_trace(go.Scatter(x=index[len(train_predictions) + self.num_prev_days:], y=test_predictions, name='Predicted Price Test'))

fig.update_layout(title=f"Stock Price Prediction for {company}",
                    xaxis_title="Time",
                    yaxis_title="Price",
                    legend_title="Legend",
                    width=800,
                    height=500)#,paper_bgcolor="gray")

fig.show()

```

The method visualizes the predicted results, simultaneously comparing to the actual stock prices. It plots the actual and predicted closing prices for both the training and testing datasets using Plotly.

vii. Example

Code :

```

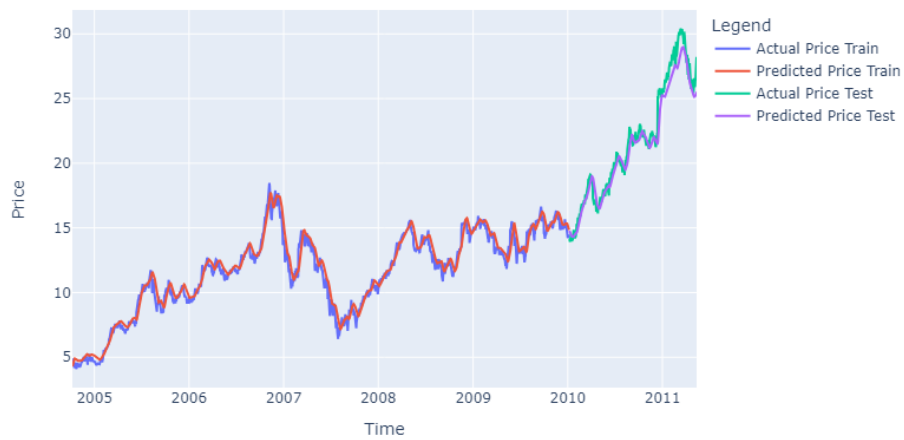
company_name = 'Alphabet'
end_date = int(pd.Timestamp("2014-05-28").timestamp())
num_prev_days = 50
name = [i for i, v in companies.items() if v == company_name][0]

finpred = Finpred(name, end_date, num_prev_days)
finpred.load_data()
finpred.build_model()
finpred.train_model()
train_preds, test_preds = finpred.predict()
finpred.plot_predictions(train_preds, test_preds, company_name)

```

Result :

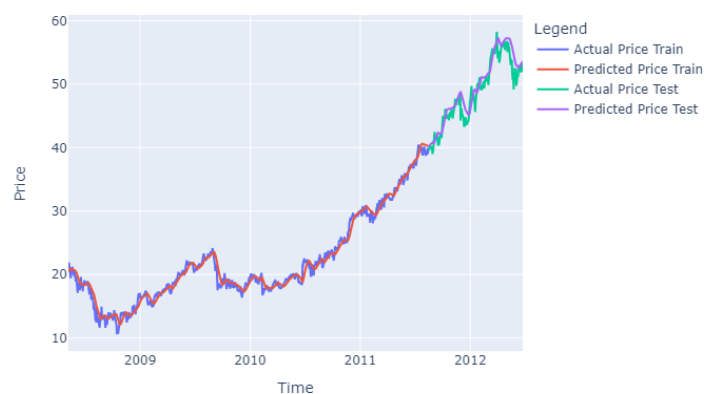
Stock Price Prediction for Alphabet



This is an example of stock price prediction for 'Alphabet'. Purple line is actual prices of the train data and red line is predicted prices of train data. Data is divided for two part around 2010 from this plot, and that is the point where plot switches to test data. Green line is actual prices of the test data and purple line is predicted prices of test data. Those two lines have a similar flow, so for this 'Alphabet' company, our model is predicting stock prices well.

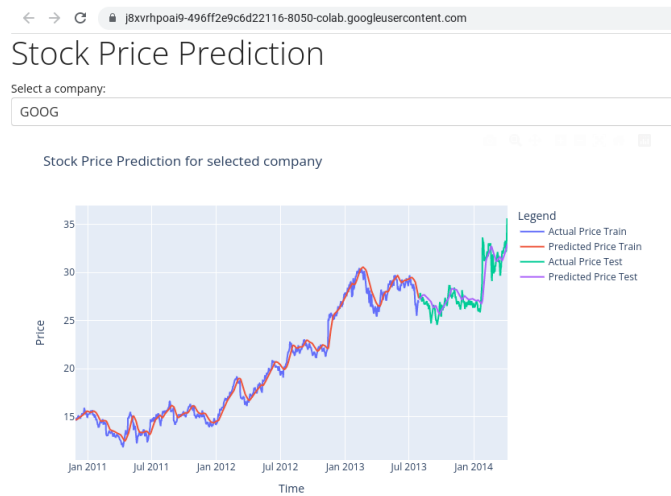
Additionally, another example for 'Visa'.

Stock Price Prediction for Visa



This example is showing a good performance for predicting stock prices as well. Line of predicted price follows the flow of the actual price line. Therefore we can know predicted prices resemble actual prices.

viii. Dashapp



We used Dashapp for the user interface. From Dashapp, a website for stock prediction is made. User can select one of the options of companies and the website shows a graph of actual and predicted stock prices for the both training and test datasets.

3. Conclusion

i. Summary

We constructed a LSTM model for stock price predictions by using historical stock price data from Yahoo Finance. Stock price data is retrieved from Yahoo Finance by URL using each different company name and end date which user wants. Dataset is created as input-output pairs form, and it is split to training dataset and testing dataset. Input represents the closing prices of the preceding days based on the specified number of previous days and output is the next day's closing price. Then we build the LSTM model using Adam optimizer and MSE loss function. With the built model, closing price predictions are calculated for both the training and testing dataset. And finally, actual and predicted closing prices are plotted in one graph so that we can simply understand the flow of stock price change and check if the predicted prices resemble actual prices.

ii. Analysis of result

From the example graph of 'Alphabet' and 'Visa', we can see a good precision of prediction that predicted values for the test dataset have similar flow with actual

values. Therefore, by our LSTM prediction model, we can well predict the stock price of a specific company generally. But as the stock market still has lots of elements which are hard to predict, the actual price for the future can be different, thus we should be aware of the instability of the market and not fully rely on the prediction. Besides that, we noticed that the predictions are delayed, namely, the model does not predict the change in price instantly, but it takes time for it to start following the actual trend of the price. However, we anticipate that this project will be useful as a reference for stock price prediction. From this result, we can further develop the prediction model by using other elements related to stock price.

4. Bibliography

- 1 Shah, D.; Isah, H.; Zulkernine, F. Stock Market Analysis: A Review and Taxonomy of Prediction Techniques. *Int. J. Financial Stud.* **2019**, *7*, 26. <https://doi.org/10.3390/ijfs7020026>
- 2 Khumprom, P.; Yodo, N. A Data-Driven Predictive Prognostic Model for Lithium-ion Batteries based on a Deep Learning Algorithm. *Energies* **2019**, *12*, 660
- 3 Hochreiter S , Schmidhuber J . "Long Short-Term Memory" Neural computation : 1735-1780

Code was based on:

<https://towardsdatascience.com/predicting-stock-prices-using-a-keras-lstm-model-4225457f0233>