Tweetpull.py

The purpose of this file and script is to collect tweets using the twitter API ‘tweepy’. The tweets that are collected are related to the query with the key words of ‘oi, wti, and gas’. In addition to this, tweets that are retweeted are excluded and thus only original tweets are collected.

Going over the libraries that are imported, pandas is for data manipulation, tweepy is for interacting with the twitter API, datetime is used to work with the date time values, time is for time related functions, while timedelta is for time calculations.

Firstly the twitter API credentials are defined. These are obtained from twitter by registering as a developer. The keys and secrets are then listed.

Next the tweepy client is initialized using the provided academic research tokens. The query is defined and the endtime is specified. The endtime in this script indicates the end date of the collector to stop searching for. In effect, this cause the tweepy API to search for tweets the previous day of the stated endtime. Next, the amount of days to iterate though is specified. For the sake of the paper and project, 4450 days was conducted. The list of tweet text and tweet dates are empty list created for the storage of the collected tweets and their dates. Request and request count are used to keep track for the number of API requests made.

At the start of the loop I iterate through each day that is required to search for tweets. Then on each day, the script makes an API request to search for the tweets. The tweets are retrieved using the client.search\_all\_tweets method, which is only allowed by having an academic researcher account. It allows all tweets from any previous date without any limits to be searched. The tweets are then extracted and appended to their text and date list respectively. Then the endtime is updated for the next day to be searched. To stay within limits of the API rate limits, the script sleeps 1.1 seconds between requests and the request is incremented to keep track of how many request are made. Once the request reaches 300, the scrip then waits 15 minutes until the next 15 minute window can be used to open request again. The logic is that per request there is a maximum of 500 results of tweets that can be shown, and that there can be a maximum of 300 requests every 15 minutes.

The data is then saved to a pandas dataframe and finally written as a csv for further processing. The results of this over 4450 days were over two million tweets.

Tweetprocessing.py

This file is used to process the csv file of tweets that were acquired from the tweetpull file. This uses pandas for data manipulation, regular expressions, re, for text cleaning and textblob for sentiment analysis.

Firstly, two dataframes are created for two csv files respectively. The first is the outputs of the previous file contain millions of tweets, while the second is daily price chart of crude oil WTI. Index col specified as date in the oil price dataframe is to ensure that the date column is used as the index. Next, the first date column is modified so that the time is not included as only the date is wanted.

The function cleantweet is used to clean the text of the tweets using regular expressions. It does this by removing mentions, hashtags, retweets and hyperlinks. This function is then applied to all of the text of each tweet thus cleaning all tweets as bare sentences and strings.

Now that the tweets are cleaned and able to be read coherently, they are fed into TextBlob to perform sentiment analysis. Of course that means that these data points and tweets are machined labeled so they may not be accurate, however it would be unfeasible to perform manual analysis on over two million tweets. Textblob performs a polarity scoring so that each tweet has its own polarity score. Then with all the tweets scored, they are combined together and grouped by dates. Each day has its average polarity scored based on the collected amount of tweets for that day. Then by manipulating the dataframes, for each date, the daily oil price and average tweet sentiment is gathered. They are combined with an outer join on date.

Oil\_with\_sentiment.py

Firstly, libraries are imported for use in this file as the purpose here is to use a Long Short-Term Memory LSTM neural network to predict oil prices based on a combination of daily close price, and sentiment analysis scores from twitter data. The libraries are numpy, pandas, sklearn, matplotlib, and tensorflow.

The seed is set for both numpy and for tensorflow. Then the final oil csv that was created in the previous file is read as a dataframe here. Index column is specified for the date, while dropping null values. After this, the specified columns of the price, polarity and twitter volumes are specified.

Next is the window\_data function. The window data function is used to prepare data for the LSTM. It generates input features with a target of y sequences from the dataframe. The parameters it takes is the dataframe, the size of the window, columns of inputs or features and the target column. In this case, the window size is 5 to represent the 5 weekdays that are trading days in the week. Secondly, the inputs of features are those that are found in the dataframe while the output target is the predicted price. Four empty list are created to store the input feature sequences and the target output sequence. The days specified in the window is looped through. Within the loop, the data from the dataframe is extracted based on the current i value which are finally appended to the empty list. At last the list of feature sequences are combined into a single numpy array using np.hstack while the target list ‘y’ is reshaped into a 2d array using np.aray().reshape(-1,1). The purpose to reshape the y is to ensure that the structure is compatible with the requirements of most machine learning libraries. It must be emphasized that the list that are created is a list of sublist and that the slicing of the data extracted from the columns are the same for the X features but for the y, the only slice is for one day. In other words with a window of 5 days, 5 days of data are generated into a list of sublist, and the corresponding y value target is just one price point based on the value of i + window instead of i: (i+window). The final corresponding information after assigning the 3 features as X and the target as y would look similar to concatenating the first 5 days information for each column as one X value, and then the 5th day’s target price.

Continuing from here, the X split and y\_split are generated as 70% of the data that was created from the dataframe. So if the dataframe is 2000 values long, since the window is 5 1995 target values would be generated alongside the concatenated x values. 70% length of this would leave 1400 values for training and the remainder for testing. Next the MinMaxScaler from scikit learn is initialized for the test values x,y and training value x,y. After this the scaler is fit onto the X training and y training data. This is to allow the scaler to know the minimum and maximum that is to be used on the data. Now the x train and y train data are scaled based on the parameters set previously found from the scaler. This is done by reassigning x train and y train as x\_train\_scaler.transform(x\_train). Likewise this is also done with the y training data as well. From here, the exact same steps are performed for the testing data as well. In other words, the steps are, initiate scaler, fit the data via the scaler. Transform the data based on the fit. The point of the data is to be within a range of 0 and 1. This makes it suitable for machine learning. Lastly, the training and testing data are reshaped to be fed into the LSTM. The LSTM expects a 3D format of samples, time steps, and features. Thus, when reshaping, the sample size is the number of windows that were created on a rolling basis. The timesteps is equal to the number of windows multiplied by the number of features. Lastly, since the features are combined horizontally the features is 1.

Now that the data is complete and ready the LSTM model is defined and built. The model is a sequential model from the keras library. Sequential allows the stacking of layers in a linear manner. The number of units, 9, represents the amount of neurons in the layers while also adding a dropout fraction of 0.2. In the first layer, we add a LSTM layer specifying the number of units as above, with the return sequences as true. Lastly, the input shape is specified by taking the second dimension of the X\_train numpy array using the shape function. Since the second dimension is the number of timesteps, .shape[1] is required.

After the input layer and first layer of the model is created, dropout is added with the second and third layer following in a similar manner to the first. However, in this case, the second and third layer does not require a input shape specification as it was already stated in the first layer. Since the first layer is the layer where the features are input, the output of the first layer should smoothly move onto the subsequent layers. Lastly, the output layer is a dense layer. The purpose of a dense layer is to have a layer that is fully connected. This means that the previous layer which is a LSTM layer with 9 nodes, has every node making a connection to the final dense layer of 1 node. 1 node is often used in time series predictions since that is the final prediction. However in classification cases the dense layer for output can have a multitude of nodes that represent each class. Lastly, the model is compiled using model.compile to configure the learning process before the training starts. The optimizer used here is an adam optimizer which is very popular. For loss, it is using a mean squared error. It measures the mean squared difference between the predication values and actual target values and attempts to lower the loss at every epoch.

The model is fit to the x\_train and y\_train data. Epochs are specified at 100 with a batch size of 5. A batch size defines the number of samples per gradient update. The model will be updated after every 5 samples that have been processed. After training, the model is evaluated using the testing data instead of retrieving the loss metric directly from the mode. Predictions are made using the x\_test data and then the mean squared error is calculated using those predicted values and the actual y\_test data. After this, the square root of this value is taken to obtain the root mean squared error. Even though the loss is mean squared error for the model is used, it is not used in the evaluation metric. Only it is used in the training.

To create predicted prices that represent the oil prices, the y test scaler data must be inverse transformed using the predicted data from the model. The logic is this: x train data is created from the window\_data function. A scaler is initialized and then fit using the x train data. Then the x train data is transformed using that fitment. Lastly, it is then reshaped for the LSTM. After training, during the testing phase, predictions are made. These predictions are predicted Y values. Using these predictions, they are then inverse transformed using a y test scaler. The final result is predicted prices that represent oil prices. Similar to this, the real prices are also inverse transformed. Now these can be compared in a realistic manner.

Lastly, the predicted and real prices are used to create a dataframe named stsocks. Here stocks is used to create a csv file as well as used to plot. The title and labels are named accordingly and using plt.plot to create a line chart, the x axis is the dates. This is specified using the stocks.index since the dataframe stocks has the dates as the index. The y values are then specified using the name of the column, ‘real’ and ‘predicted’ respectively.