Sentiment Analysis on Yahoo Finance Conversations

Mark McCauley

Introduction:

News and posts about stocks are often flooded with conflicting opinions and valuable articles can take a lot of time to read. It can sometimes be quite difficult for an investor to obtain an accurate reading of the market sentiment. Figuring out why investors buy and sell stocks is an extremely complex problem with a lot of variables. There is a huge amount of data available to us about the stock market, but most people require financial advisers to help them make decisions on their investments. Deep learning is opening the door to give investors extremely valuable insight about financial markets through analyzing this data. One area in particular that deep learning is excelling is sentiment analysis by using long short-term memory network models. Through this paper, I hope to show that deep learning is able to give investors valuable insights on market sentiment which they may not be able to obtain through a financial advisory institution.

Method:

Collecting Data:

To perform a sentiment analysis, I knew it would require a significant amount of data. Yahoo Finance contains a plethora of financial information and seemed like the obvious place to experiment with creating a model. There is a section for each stock on Yahoo Finance called “Conversations” where users are able to post their own opinions on stocks and interact with other users. I decided this section would be a solid place to start experimenting since it was not an excessive amount of data, but it was significant. To collect the necessary data, a data crawling script was written to collect at least the first 20 conversations for each stock on the NYSE. Selenium and Beautiful Soup were used together in order to pull this data. The main problem I encountered with this data was finding a dictionary with reasonable sentiment ratings to accommodate my this data. I was able to find dictionaries containing financial words and ratings, but was unsuccessful in finding words and phrases together. I considered generating my own dictionary. While this could provide better results, it would be costly and challenging.

```
for ticker in tickers:
    try:
        #ticker = ticker[0]
        print("Processing ticker: " + ticker +"\n")
        Company_Dict = {}
        Company_Dict["Ticker"] = ticker

        r  = requests.get("https://finance.yahoo.com/quote/"+ticker+"/profile?p="+ticker)
        data = r.text
        soup = BeautifulSoup(data, features = "html5lib")

        company_name = soup.find(attrs = {"class" : "Fz(m) Mb(10px)"})
        Company_Dict["Name"] = company_name.get_text()

        r = requests.get("https://finance.yahoo.com/quote/"+ticker+"/community/")
        data = r.text
        soup = BeautifulSoup(data, features = "html5lib")

        comments = []

        body = soup.find_all(attrs = {'class' : 'Fz(14px)'})
        for b in body:
            line = b.get_text()
            if len(line) > 20:
                comments.append(line)

        if len(comments) < 1:
            print "No Conversations for:" + ticker
        else:
            Company_Dict["Conversations"] = comments
            with open(Company_Dict["Ticker"]+".json", 'w') as data_file:
                json_string = json.dump(Company_Dict, data_file, indent = 2)

    except:
        print("Yahoo Does not have ticker: " + ticker)
```

Once I had this data, I knew I would need some type of dictionary of financial terms with respective sentiment ratings. For this I used the NTUSD-Fin dictionary and scaled the sentiment ratings to fit to a 0-1 scale.

```
with open('NTUSD_Fin_word_v1.0.json') as data:
    data = json.load(data)
    sents = {}
    for d in data:
        if d['market_sentiment'] < 0:
            sents[d['token']] = d['market_sentiment'] / -250
        else:
            sents[d['token']] = d['market_sentiment'] / 1.224

    sortedsent = collections.OrderedDict(sents)

    out = open('words_w_scores.txt', 'w')
    out.write('word|sentiment\n')
    for i in sortedsent:
        out.write(i.encode('utf-8') + ' | ' + str(sortedsent[i]) + '\n')

    out.close()
```

Preprocessing:

I then reprocessed the data I collected from each stock to make it easier for the model to read. The main idea is to predict a sentiment rating on each conversation for a stock then combine these and assign the average value to the sentiment rating of the stock in general. The NLTK library was useful for effective preprocessing. The mean number of words in each conversation was about 54 words so I cut off each accordingly to make the prediction process smoother. Each preprocessed stock was stored in a common folder.

```
documents = sorted(glob.glob("/home/mark/Research/data/stocks/*.json"))

lens = []
for doc in documents:
    preprocessed = []

    try:
        with open(doc) as data:
            data = json.load(data)
            convos = data['Conversations']

        for c in convos:
            processed = ''.join(i for i in c if i not in string.punctuation)
            processed = processed.split()
            processed = ' '.join(i for i in processed if i not in stop_words)
            processed = processed.lower()
            processed = processed.split()
            processed = processed[0:60]
            processed = ' '.join(i for i in processed if not i.isdigit())
            preprocessed.append(processed)
```

Embeddings:

To create my word embeddings, I used the pre-trained GloVe embedding model. I used a lookup on the pre-trained vectors to replace each of my words with its respective embedding.

```
def pipeline(data, word_index, weight_matrix):
    rows = len(data)
    ids = numpy.zeros((rows, 1), dtype='int32')

    word_index = {k.lower(): v for k, v in word_index.items()}
    i = 0

    for index, row in data.iterrows():
        word = row['word']
        word = word.strip()
        try:
            ids[i][0] = word_index[word]
        except Exception as e:
            if str(e) == word:
                ids[i][0] = 0
            continue
        i += 1

    return ids
```

Model and Training:

After splitting the data into training, testing, and validation sets. To help create my model, I used 4 different layers:

- An embedding layer with a vector size of 100 and a max length of 1 since our dictionary only contains words and not phrases.
- 256 bi-directional LSTM layers with a dropout to prevent overfitting.
- A 128 layer dense network which that takes input from the LSTM layers. There is a dropout of 0.8 here
- A 10 layer dense network with softmax activation.

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1, 100)            40000100
_________________________________________________________________
bidirectional_1 (Bidirection (None, 256)               234496
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 40,268,782
Trainable params: 268,682
Non-trainable params: 40,000,100
_________________________________________________________________
```

```
model = Sequential()
model.add(Embedding(len(weight_matrix), 100, weights=[weight_matrix], input_length=1, trainable=False))
model.add(Bidirectional(LSTM(128, dropout=0.2, recurrent_dropout=0.2)))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.8))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print model.summary()

best_model = keras.callbacks.ModelCheckpoint(data_dir + '/best.hdf5', monitor='val_acc', verbose=0, save_best_only=True, save_weights_only=False, mode='auto', period=1)

model.fit(train_x, train_y, batch_size=5, epochs=10, validation_data=(val_x, val_y), callbacks=[best_model])
```

I ran the training with a batch size of 5 items at a time since the size of the dictionary was relatively small. The training was set to run 10 epochs. After training, the model provided a result of ~35% accuracy which is not terribly impressive.

Predictions:

I ran a live test on all of the stock data that had been preprocessed and stored in the common directory. Each result was stored in a text file called “sentiment_ratings_words.txt” presenting in “Ticker | Sentiment” format. The sentiment assigned to each ticker was the average of all the sentiments of each conversation posted about the stock. The sentiment is on the 0-1 scale, close to 1 is a positive rating, close to 0 is a negative rating, and close to .5 is a neutral rating.

The results of these predictions were not really what I was looking for since almost every stock was given a positive sentiment. A neutral rating was considered to be a sentiment between 0.45 and 0.55

Results:

Mean: 0.64	Positive Stocks: 47178
High: 0.84	Negative Stocks: 43
Low: 0.3	Neutral Stocks: 7231

Improvements:

Since the dictionary I used only contained words and their respective sentiments, I do not believe my model was as accurate as it could have been and was limited by this fact. Therefore, I decided to train a model with a general non-finance oriented dictionary with multiple phrases. My theory was that this would provide a better trained network and would be able to give me the results I desired.

For this I used the Stanford sentiment tree bank data which is much larger than the dictionary I previously used. Training my model took much longer, but I seemed to generate better results containing mostly neutral sentiment. With the new model I was able to achieve ~46% and the results for each stock can be seen in “sentiment_ratings_phrases.txt”.

Results:

Mean: 0.45	Positive Stocks: 9634
Max: 0.73	Negative Stocks: 25464
Min: 0.2	Neutral Stocks: 12846

These results are clearly better distributed and are much closer to what I was looking for. The data seems to be skewed to the negative stocks, but it is not clear to what would be causing this.

Conclusions:

When looking at the sentiment rating of a stock then looking at the conversations, it seems that the ratings are fairly reasonable for my second experiment. The model is constricted to the dictionary available and if I were able to train it with data containing phrases specific to finance I would be able to obtain better results. While I would not invest in a stock based on these sentiment ratings, I believe this is a step forward to generating something better and I wish to continue experimenting with sentiment analysis on financial data.

My next step is to develop a model to predict the sentiment of 10-Ks because this is some of the most relevant financial data available and can be difficult for most people to interpret. A model capable of this will be much more complex, but I am up for the challenge.

References:

1. Prajwal Shreyas - https://gitlab.com/praj88/deepsentiment
2. NTUSD-Fin dictionary - http://nlg.csie.ntu.edu.tw/nlpresource/NTUSD-Fin/
3. GloVe Embedding - https://nlp.stanford.edu/projects/glove/
4. Stanford sentiment treebank - https://nlp.stanford.edu/sentiment/treebank.html