Word Embeddings for Identifying Company Names and Stock Tickers from Social Media Posts

Brian Ward
ward.bri@northeastern.edu

Abstract

This project aims to create a tool which can be used to extract possible market insights on companies which are discussed on social media platforms such as Reddit. The main goal is to identify company names/tickers from raw text. Assuming that these company names would be used in similar context, word embeddings which are trained based on context were used to identify these targets. Using this approach I was able to extract company names from nearly 300k of the ~1.16 million reddit posts consisting of over 800 unique values. Testing on a 1000 record ground truth this model was able to achieve 87% accuracy. The second part of this project aimed to classify reddit posts into one of the following three sentiments: negative, neutral, or positive. I was able to achieve ~70% accuracy for this task by use of naive bayes classification.

1 Introduction

Over recent years the trend for younger, less experienced individuals to take part in the stock market has sky rocketed with nearly 10 million new retail investors in 2020 [10]. The introduction of easy to use investment apps such as Robinhood and zero commission trades have greatly reduced the barrier to entry for trading. The culmination of this trend was seen when a community of traders on a Reddit message board, r/wallstreetbets, conducted a short squeeze on GameStop ($GME) in January 2021. At a certain point $GME was up over 2000% YTD, due to this online community orchestrating this plan. Although the total losses of the investment funds which were shorting game stop is unclear it is likely to be in billions [11]. This was an eye opening event for wall street and definitely left a clear message that these social media traders were not insignificant.

It is certainly not a new idea to track social trends to look for associations in the stock market, but it is now evidently a necessary step. This notion and my interest in NLP was my motivation for trying to extract insight from social media posts such as Reddit. To focus this idea a bit more I chose to perform two tasks on a given social media post :

1. Identify company names and/or stock tickers
2. Perform sentiment analysis on those posts

I sourced ~1.16 million posts from r/wallstreetbets from Kaggle [5][6] for creating the tool to identify company names/tickers. After processing this data I used the Gensim library [12] to create word embeddings for the vocabulary extracted from the reddit posts. Word embeddings are vector representations of words which are learned based on a words given context. Under the assumption that company names and stock tickers would be used in similar context they should also have similar word embeddings. These word embeddings were then used to identify over 800 unique company names/tickers from the input data.

Lastly Naive Bayes was used to classify each reddit post into a sentiment category of either negative, neutral, or positive. Three different Naive Bayes methods from Scikit Learn [7] were compared including:

1. Multinomial Naive Bayes
2. Bernoulli Naive Bayes
3. Complement Naive Bayes

A dictionary based approach was also considered using NLTK's Vader [8] sentiment analysis tools.

The remaining sections of this paper are organized as follows: 2 background on the methodology used, 3 detailed project description, 4 experiments conducted, 5 related work, and 6 Conclusion.

2 Background

The implementation of this project is strongly dependent on the idea of word embeddings or vector representations of words. Representing words as vectors is a fundamental tool in natural language processing as it gives us a way to represent a word numerically. The most basic form of vector representations is done with one hot encoding.

One hot encoding of a word in a vocabulary represents a given word as a vector equal in length to the size of the vocabulary where each number in the vector maps to a word in the vocabulary. If the word a given number maps to is the word we are representing then the value is 1, otherwise it is zero.

```
Vocabulary :
     ['do', 'you', 'like', 'green', 'eggs' 'and', 'ham',
      'would', 'them', 'here', 'or', 'there']
```

	Vocabulary							
	do	you	like	green	eggs	and	...	there
eggs	0	0	0	0	1	0	...	0
you	0	1	0	0	0	0	...	0
there	0	0	0	0	0	0	...	1

```
eggs = [0,0,0,0,1,0,0,0,0,0,0,0]
```

Figure 1: One hot encoding example.

Figure 1 gives an example of how the word 'eggs' would be represented as a one hot encoding vector. There are some clear downfalls to representing words in this manner. One hot encodings are generally extremely large and sparse vectors. They also do not hold any other information about the word it is representing other than the word itself. In other words two similar words 'good' and 'great' will be just as similar to each other as 'good' and 'hippopotamus'.

Enter word embeddings; short and dense vector representations which store actual information about the word they represent. Word embeddings were introduced by the distributional hypothesis : words that occur in similar contexts tend to have similar meanings [1]. In the next section I will discuss how context of a word is used to learn these embeddings for all the words in a documents' vocabulary.

3 Project description

3.1 Data input and pre-processing

This project makes use of two large datasets [5][6] holding posts from r/wallstreetbets. Between these two datasets we have reddit posts and comments from as far back as 2012 to the present. For the training of the model any posts were split into sentences resulting with a total of ~1.4 million sentences from ~1.16 original records.

Pre-processing the raw text is an important step to remove noise and ensure everything is uniformly formatted for the model to perform to correctly. There are many factors to consider :

- Letter case
    - Made everything lowercase
- Punctuation
    - Removed all punctuation except apostrophe.
- Numbers
    - Removed all numbers
- Other non alpha-numerical characters (e.g. emojis)
    - Removed Emojis
- Stemming
    - Chose not to stem words
- Stop words
    - Chose not to remove stop words
- N-gram
    - Will test results on 1,2,3-gram (more in Experiments section)

The above string processing steps were chosen after some trial and error and viewing the testing results (more in Experiments section). These data-sources are now processed and ready to be input into our Word2Vec model.

3.2 Gensim word2vec

The main algorithm for this project is a word embedding algorithm commonly referred to as word2vec from the Gensim library [12]. Word2vec is an algorithm designed to create word embeddings or vector representations of words from a given set of documents. Word2vec creates short dense embeddings, apposed to the sparse embeddings we touched upon in in the previous section.

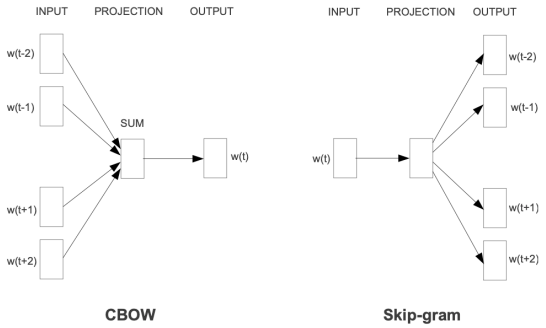

Figure 2: CBOW vs Skip-gram [3]

These word embeddings are usually created using one of two algorithms: continuous bag of words (CBOW) or skip-gram with negative sampling (SGNS). The former attempts to predict the target word from its context words, where the latter attempts to predict the context words from the target word as depicted in figure 2. CBOW is a quicker algorithm but does not perform as well with rare or infrequent words. For this reason I chose to select skip-gram for this project. For the remainder of this paper we will focus on the skip gram with negative sampling algorithm.

3.3 Skip gram with negative sampling

Let's first discuss the input data for the skip gram algorithm. Word2vec will take in a pre-processed document(s) of sentences, where a sentence is a list of the words or tokens in the sentence, and the document is a list of these sentences. The vocabulary of the model will consist of every unique word in the document.

Let's now look into the classification task in more detail.

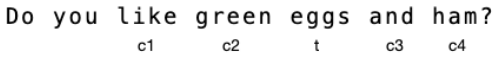

Figure 3 : a sentence example showing the target word and its context words with a window size of 2.

Figure 3 gives an example sentence that skip gram might encounter. The immediate goal is to compute the probability that a given word **c** is a real context word of the target word **t :**

$$P(+|t,c)$$

This is the fundamental probability equation on which skip gram is built [1]. How is this probability actually computed? One important thing to note here is that a skip gram model stores two vectors for each word in its vocabulary. One which represents the word as a target word and one which represents the word as a context word. Using this simple tactic, skip gram assumes that a target word vector should be similar to its context word vectors. This assumption lets skip gram determine the probability of a context word **c** being a real context word of the target word **t** by comparing their context and target vectors respectively.

$$Similarity(t,c) \approx t \bullet c$$

$$t \bullet c = \sum_{i=1}^{n} a_i b_i$$

This relies on the assumption : two vectors having a high dot product are similar (as cosine is a normalized dot product) [1]. Of course this is not going to give us a probability as the bounds for possible values are - ∞ to ∞. We will now make use of the logistic function to turn this number into probability with a value ranging 0.0 - 1.0.

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

This equation represents the probability of a target word and one context word, but depending on the window size and target word location, there can be multiple context words we'll call **L**. Skip gram assumes these multiple context words are independent, allowing for the multiplication of these probabilities simplifying the equation to:

$$P(+|t,c_{1:L}) = \prod_{i=1}^{L} \sigma(-c_i \bullet t)$$

This final equation describes how we can calculate the probability the target word has the $\mathbf{c_{1:L}}$ context words based on the similarity of the target vector of word **t** and the context vectors of words $\mathbf{c_{1:L}}$ [1]. Clearly these probabilities depend on good vector representations of the vocabulary which we have not yet discussed.

3.4 Learning embeddings in skip gram

As previously mentioned this word2vec model will create two vectors per word in the models vocabulary. One for the word as a target and one for the word as a context. How are these vectors actually learned? This learning process can be broken into three high level steps:

Step 1: randomly assign embeddings.

Step 2: iterate over document shifting target embeddings and corresponding context embeddings closer together.

Step 3: for the number of context words L of target word **t**, randomly select L*k (k is a parameter) negative samples. Shift embeddings of target word **t** and context embeddings of negatively sampled words further apart.

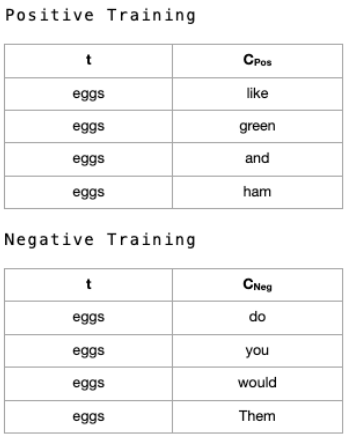

Figure 4 : learning skip gram embeddings with negative sampling (k=1)

In figure 4, we can see that we have the same number of negative samples as we do positive samples or 2L, (k = 1). this is for simplicity reasons, as k is a customizable parameter but is usually greater than 1 leading to a much higher negative sampling than positive sampling [1].

3.5 Choosing negative samples

In order to have good negative samples we want to ensure that less common words in the vocabulary are still sampled sufficiently. To achieve this we sample words according to their weighted unigram frequency :

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

We can see how using this weighted distribution can help increase the rate at which we sample infrequent words in the vocabulary. For example let's take two words, a and b, which have frequencies P(a) = 0.99 and P(b) = 0.01 and

observe their new distribution using alpha = 0.75, which is common practice [1].

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

We can see that infrequent words are given a slight boost in sampling using this technique (example sourced from Jurafsky et al. [1]). One more thing to note here is that negatively sampled words are only constrained to not be the target word, so it is possible to negatively sample a word which could be a context word of **t** elsewhere in the document. Given the size of training data and its random sampling however this possibility becomes negligible.

3. 4 Gradient descent

The main goal of this algorithm can be summed up in two parts :

1. Maximize embedding similarities of target words and positive context words.
2. Minimize embedding similarities of target words and negative context words.

These goals can be used to create the following loss function:

$$Loss = -log[P(+|t, c_{pos}) \prod_{i=1}^{k} P(-|t, c_{neg})]$$

This loss function can then be minimized with stochastic gradient descent [1].

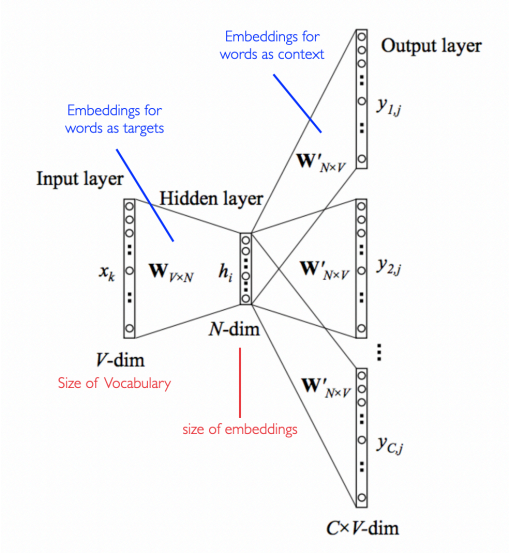

Figure 5: NN representation of skip-gram [3]

Figure 5 shows the neural net depiction of an iteration of the skip gram algorithm. The input layer takes in the target word as a one-hot-encoding vector. The size of the hidden layer will be the embedding length, which is a hyper parameter of the skip gram algorithm (more on this in next section). The first weight matrix, W_{VxN} , represents each word embedding for every word in the vocabulary. These embeddings represent the words as target words.

The hidden layer (sometimes referred to as the projection layer) represents the transposed word embedding that corresponds to the target word (one-hot-encoding input layer). Essentially this just gives us the embedding for the target word, hence its dimension is embedding length x 1.

The weight matrix, W'_{NxV} , depicts the word embeddings for each word in the vocabulary as context words. The output layer represents a probability distribution of each word in the vocabulary being a context word of the input target word. Each separate output layer represents a different context word for the target word we are currently looking at, (C : number of context words for target word). These are calculated using the softmax activation function [9]. Again, this is repeated as we iterate over the document using gradient descent to learn both sets of these embeddings.

3.5 Extracting companies using word embeddings

Once we have successfully created word embeddings for our vocabulary we can get back to our main goal of identifying our company names and stock tickers. We now have embeddings which in theory should represent their related words to the context in which they are used. Under this assumption company names and stock tickers should all have some similarity in their word embeddings. The goal will be to find a representative or average vector which will be similar to all of our target words (company names and tickers) and dissimilar to the other words in the vocabulary. Once we have a representative vector we can use it to identify company names by calculating the cosine similarity of a given words embedding and our representative vector. Words with a cosine similarity greater than some pre-determined threshold, say 6.5, can then be assumed to be a company name or ticker.

3.6 Sentiment analysis with naive bayes

The next goal of this project was to be able to give a sentiment score to these reddit posts. Sentiment analysis is a very popular NLP process which can be used for a wide variety of insights such as tracking twitter sentiment for political candidates throughout a campaign. The motive for sentiment analysis for this project is to allow for the tracking of sentiment for different stocks across reddit and possibly other social media platforms.

There are many different methods for sentiment analysis that range from very basic to advanced. A basic method would be to have a dictionary of words all which have

some pre-determined sentiment (negative, neutral, positive) and using that dictionary to sum up the sentiment of a collection of words. State-of-the-art NLP tools such as BERT (Bidirectional Encoder Representations from Transformers) can also be used for sentiment analysis and would likely do a much better job for such unstructured text as reddit posts. Unfortunately due to time constraints and issues with getting TensorFlow to work on my computers architecture I chose not to explore BERT for sentiment analysis. Instead I chose to explore the naive bayes approach. Naive bayes can be used for sentiment analysis by classifying a set of words into one of three categories:

1. Positive [1]
2. Neutral [0]
3. Negative [-1]

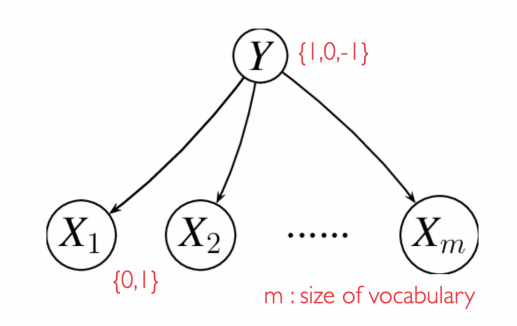

Figure 6 : Naive Bayes for sentiment classification

The features x_1- x_m represent each word in the vocabulary, where a set of words, or a document, will have either a 1 or 0 for that feature depending on whether or not the corresponding word exists in the document.

The input for a Naive Bayes sentiment classification algorithm is going to be a document term matrix (DTM). Each column in the DTM will represent a word in the total vocabulary, and each record or row will represent a document or in this case a reddit post. Each cell will be the count that the word appears in that document.

The probabilities of each feature given the three different sentiment classes can be learned or calculated by the training data. This naive bayes model assumes independence of each of the features and we are able to use the following equation to calculate the probabilities of each of the three sentiment classes:

$$P(Y, x_1, \ldots, X_m) = P(Y)\prod_i P(F_i, Y)$$

The class with the highest probability can then be chosen for that given document or post.

4. Experiments

Word embeddings are designed to capture a words meaning based on the context words found around it in the given training data. Objectively evaluating the success of word embeddings can be hard to do. There are tests that can be be done using pairs of words which are known synonyms or that can be decided to likely be used in the same context. However, when embeddings are used for a particular task, it is better to try to create a test for the embeddings ability to perform at that task. I chose to create a ground truth on which I could test the embeddings on their specific task.

4.1 Ground truth

To create the ground truth I started by randomly sampling 1000 reddit posts from the input data from r/wallstreetbets. I then read through each of them and by hand extracted any company names or stock tickers that I was able to pick out. To ensure that a stock ticker was in fact referencing a company I would type in the letters into yahoo finance to check if it was a legitimate stock ticker.

post	expected
united knows i have fomo problems and is exploiting my disorder	united

Figure 7: Example ground truth entry.

To test against the ground truth, instead of testing each post individually and looking at the success rate for all 1000 posts, I chose to test all the words individually. This would help remove the seemingly increased accuracy, based off the ability for the model to correctly identify some of the most popular companies (e.g. ‘gme’). Instead I created a set of all the words in the 1000 posts and then created a set of all the expected company names for those posts. I could then use these sets to test the model against so that I am only testing each expected word once and each unexpected word once. I calculated the scores as follows:

When run just on the expected set:

$$expectedScore = \frac{|extracted|}{|expected|}$$

When run on un-expected set:

$$Unexpected = Vocab - Expected$$

$$unexpectedScore = \frac{|unexpected| - |extracted|}{|unexpected|}$$

$$totalScore = mean(scoreExpected, scoreUnexpected)$$

I believe this gives a more objective score of how the embeddings are able to differentiate between company names/ tickers and other words in the vocabulary.

4.2 Representative vector

The first step in the process was to create the representative vector which will be used to compare to the other words. I chose not to focus on coming up with a systematic method of creating a perfect representative vector. Instead I hand selected embeddings to average until I found results I was happy with.

```
myVec = meanVector(wv, positive=['gme', 'amc', 'tsla' 'appl', 'microsoft', 'facebook', 'home_depot',],
                       negative=['hoooooold','boyssss', 'gooooooooo', 'babyyyyy'])
```

Figure 8: Example representative vector

After many attempts and by looking at the top 500 most similar words in the vocabulary and the accuracy of the model I was able to create a good representative vector.

4.3 Parameter selection

The Word2Vec model training does have a few hyper parameters that I was interested in testing systematically. I chose to focus on 4 different parameters:

1. Similarity Threshold
2. N-Gram
3. Window Size
4. Vector Size

Similarity threshold

Let's first discuss the similarity threshold as it will have a large effect on the scoring of the models. The similarity threshold is the threshold that the similarity score between any given word and our representative vector needs to be for that word to be extracted. So a higher threshold of let's say 0.95 would mean that we would extract much fewer words and would have a better unexpectedScore but a worse expectedScore. A lower threshold would have the opposite effect. Finding the right balance is of course the tricky part.

```
wv.most_similar(positive=[myVec], topn=500)[475:]
```

```
[('joystick', 0.532387912273407),
 ('afrm', 0.5323343276977539),
 ('doordash', 0.5322819352149963),
 ('arkk', 0.5318436622619629),
 ('borland', 0.5316657423973083),
 ('motherboards', 0.5314918756484985),
 ('autodesk', 0.5314642190933228),
 ('funko', 0.531302273273468),
 ('chipset', 0.5312930941581726),
 ('castor_maritime', 0.5307868123054504),
 ('ostk', 0.5307446122169495),
 ('floppies', 0.5306878089904785),
 ('omer', 0.530500590801239),
 ('itanium', 0.530326247215271),
 ('adidas', 0.5301930830834198),
 ('aws', 0.5301036238670349),
 ('ford_f', 0.529857873916626),
 ('lowes', 0.5294471979141235),
 ('tiktok', 0.5293790102005005),
 ('cvna', 0.5293539762496948),
 ('xlf', 0.5291469693183899),
 ('hd', 0.5287479758262634),
 ('put_credit_spread', 0.5287005305290222),
 ('powerpc', 0.5286581516265869),
 ('azure', 0.5280701518058777)]
```

Figure 9: Tail of top 500 most similar words in the vocabulary.

When looking at the tail of the top 500 most similar words in the dictionary to the representative vector, figure 9, It is clear to see that the similarity values get pretty small ~ 0.53 while most of the words still appear to be correctly identified. This means that a lower similarity threshold is likely to give us the best overall results. An important thing to note here is that it would likely be worse to miss out on a company or stock ticker than to falsely extract a word as a company. In other words, we should favor false positives over false negatives. For this reason I am comfortable with these lower similarity values.

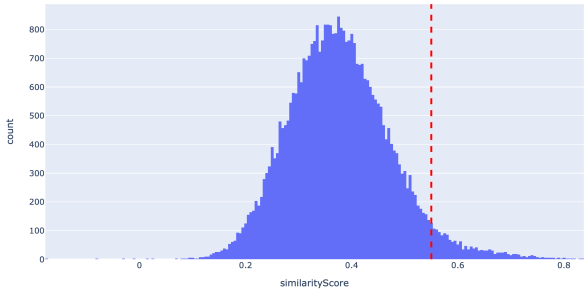

Figure 10: Distribution of rep. vector's similarity scores to the rest of the vocabulary.

The above figure shows the distribution of similarity values for the representative vector and each word in the models vocabulary. The red line indicates the similarity threshold of 0.55. This helps give us a good idea about ratio of words we would actually identify given that threshold. After some trial and error I chose to consider 4 different thresholds for these tests. I also chose to include the graphs for each of these four thresholds as I compare the other parameters as these parameters might have different effects at different thresholds.

N-gram

Although N-Gram is not a parameter of the word2vec model it is an important step in the pre-processing of the data fed into the model, and can have a great effect on the tasks we are trying to achieve. I chose to consider three options for N-Gram:

N-Gram = {single, bigrams, trigrams}

Note: all bigrams and trigrams are considered only if they are found with the minimum count = 25.

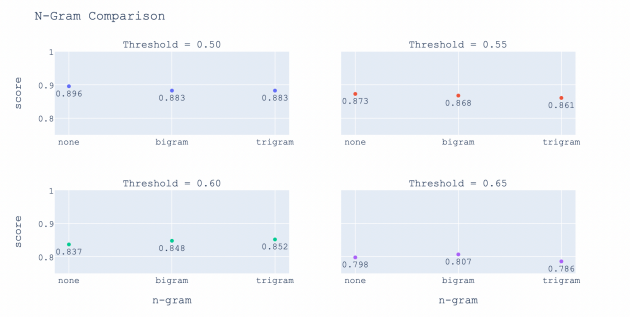

Figure 11: N-gram comparisons across the four thresholds.

We see in figure 11 that we are getting different trends across the different thresholds. I chose to continue with bigrams as it is both the middle ground and It allows us to keep company names together such as ‘home_depot’.

Window size

Window size is a parameter of the Word2vec model and specifies the number of words to the left and right of the target word that we will consider a context word. Figure 3 for example shows target word, ‘eggs’, and two target words to its left and two target words to its right. This would represent a window size = 2. This can play a large factor on the embeddings especially with less structured text such as reddit posts. For this parameter we will consider 5 different options.

$$\text{Window} = \{1,2,3,4,5\}$$

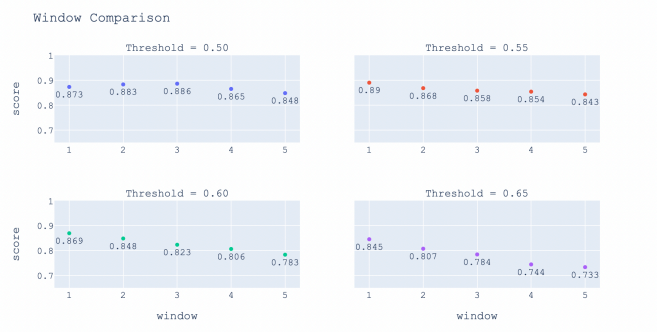

Figure 12: Window size comparison across the 4 thresholds

Here we see some pretty clear trends with smaller windows performing much better. This is likely due to the short form and unstructured format of a reddit post.

Vector Size

Vector size is another parameter of the word2vec model. Vector size is the length or dimensionality of the word embeddings that we are creating. Choosing the correct vector length really needs to be done by testing. We will consider 4 different options.

$$\text{Vector Length} = \{100,200,300,400\}$$

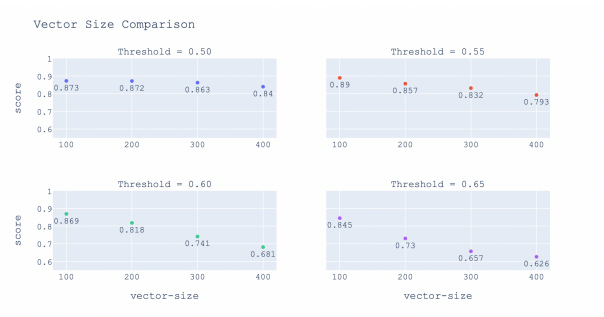

Figure 13: Vector size comparisons across the four thresholds.

Again, we see a very clear trend here showing that the smaller vector size of 100 outperformed the others.

4.4 Model results

Once determining the best parameters (bigrams, window=1, vector-size=100, threshold=5.5) we were able to achieve an overall accuracy score of 89% on our ground truth.

When applied to our original dataset of reddit posts with a threshold = 0.6 this method was able to extract companies from nearly 300k posts. At this same threshold 800 unique companies/tickers were extracted.

```
'uber': 1192,           'palantir': 1357,
 'gme': 110624,          'pltr': 6793,
 'bb': 16755,            'bbby': 960,
 'amc': 48274,           'amazon': 2925,
 'fb': 2279,             'pins': 191,
 'nok': 13880,           'pypl': 278,
 'gamestop': 12278,      'jd': 742,
 'blackberry': 1797,     'nflx': 1143,
 'tesla': 9844,          'ibm': 250,
 'tsla': 11938,          'cisco': 97,
 'facebook': 1538,       'comcast': 106,
 'tencent': 190,         'ge': 1347,
 'chewy': 280,           'tiktok': 591,
 'nokia': 3565,          'irbt': 126,
 'netflix': 1266,        'amd': 7931,
 'microsoft': 993,       'nio': 3591,
 'apple': 3604,          'snap': 2599,
 'baidu': 128,          'bed_bath_beyond': 32,
 'hp': 83,               'spotify': 281,
```

Figure 14: subset of extracted company names and their counts.

Figure 14 shows a sample of some of the unique companies and their counts found in the initial extraction of the original reddit data. The threshold of course has a large effect on the number and quality of companies that are extracted. I believe the threshold should be a parameter of this tool as it would allow a user to adjust this value for their needs.

4.5 Sentiment analysis with naive bayes

Our last step is to work on giving sentiment to each of the reddit posts that we were able to extract companies from. As mentioned in the project description section we will be using naive bayes as a sentiment classifier. The data used

for training and testing consists of both twitter and reddit text along with their sentiment {-1,0,1}. This consists of ~200k records and was also sourced from Kaggle [4]. For naive bayes implementation I chose to use Scikit Learn [7] as it offers easy-to-use training and testing for naive bayes models. Scikit Learn also makes it easy to test different versions of naive bayes. We will compare the following naive bayes implementations :

1. Multinomial Naive Bayes (MNB)
   - For multinomially distributed data [7].
2. Bernoulli Naive Bayes (BNB)
   - For data that is distributed according to multivariate Bernoulli distributions [7].
3. Complement Naive Bayes (CNB)
   - Uses statistics from the complement of each class to compute the model's weights [7].

We will also compare these naive bayes methods to a dictionary based method using NLTK's Vader (Valence Aware Dictionary for Sentiment Reasoning) [8] tool. This works using a dictionary of terms which have been pre-labeled with a sentiment score. A sentence's sentiment is then determined by averaging the sentiment scores for all of its words. This is also a naive approach as all words in the sentence are treated independently.

The first step in this testing process is to split the data into training and testing partitions. We will randomly partition 25% (~50k) of the data to be used for testing. We then train our models using the remaining training data. And finally we can test our models for accuracy.

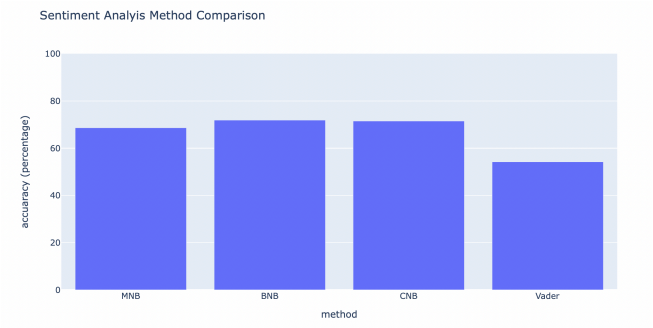

Figure 15 : Comparison of sentiment analysis accuracy for Multinomial Naive Bayes, Bernoulli Naive Bayes, Compliment Naive Bayes, and Vader Sentiment Analysis.

As we can see in figure 15 both BNB and CNB scored the highest, both around 70% accuracy. All of the naive bayes Approaches tested well above the vader approach which came in at 54% accuracy.

5 Related work

The methods used in this project were chosen not only for the best possible results, but also to fit my interests and the project scope. There are certainly many methods for identifying company names or stock tickers from natural text. One of the most basic but certainly effective methods for identifying company names and tickers would be to use a library of company names and stock tickers which a word could be tested against to determine if it was a target. Even a simple regex search of these values in a given string would likely yield pretty good results. The one issue with this approach and the main reason why I chose to pursue the word embedding methodology is that it makes certain assumptions about the text. For example in the world of social media or reddit it is very likely that people could have a nickname or a slang term for a company. This type of unpredictable text is where this word embedding approach would shine.

Looking forward there are much more advanced methods which could have been used for both the company identifying and sentiment analysis steps. The classical word2vec model still has many short-falls, For example there are many words in the English dictionary which can be spelled the same but have different or multiple meanings (called polysemy). Embeddings from Language Models (ELMo) is a much more advanced version of word2vec which can solve for the issue of polysemy as it creates word embeddings for a word as a function of the entire sentence containing that word. This gives the ability for a word to have a different vector representation under different context. ELMo is just one example of more advanced Embedding models which could have an affect on the ability to identify company names.

BERT (Bidirectional Encoder Representations from Transformers) is, according to my limited research, the state-of-the-art tool for many NLP tasks including classification. I would have loved to explored BERT for this project, but unfortunately ran into issues getting tensor flow to run on my computers architecture. BERT is also a much more advanced algorithm than word2vec and I thought it would be better to focus on an algorithm which was based on concepts covered in our class.

Good sentiment analysis is also a very hard problem. Humans talk in ways such as sarcasm, or double negatives, which are really hard for machines to pick up on. Deep convoluted neural networks have been used to try to pick up on some of the nuances of these language patterns. The naive bayes approach of assuming independence does not sound like a good approach for sentiment as words can greatly change based on their context. I chose to stick with naive bayes for this project as I was really focusing on the word2vec algorithm and again wanted to keep the scope of the algorithms within the material we had covered in class. Looking forward, I would love to explore BERT for both of the tasks I worked on for this project.

6 Conclusion

In this project my goal was to create a tool to take in raw text from a social media post, identify any company name or stock ticker, and then determine the posts sentiment.

By implementing a Gensim word2vec model to learn word embeddings on the vocabulary I was able to identify company names with an accuracy of nearly 90%. I was very happy with these results, but know there is still a lot of room for improvement. One of the areas I would have liked to explore more was the selection of the representative vector. I believe there could have been a systematic approach to learn the best representative vector to achieve a better performance.

I was also able to classify sentiment of these posts by implementing and comparing different methods of naive bayes (multinomial, bernoulli, complement) as well as a dictionary based approach. Although I am not too happy with the accuracy (~70%) of the naive bayes approach to sentiment analysis, this project definitely opened up my eyes to all of the more advanced approaches to some of these NLP tasks. I am definitely planing on exploring some of these tools in the future.

	company	count	score	meanScore
176	advanced_micro_devices	39	39	1.000000
374	planet_fitness	50	46	0.920000
225	wwe	147	134	0.911565
186	azure	36	32	0.888889
188	aws	147	130	0.884354
249	plug_power	125	110	0.880000
172	panasonic	32	28	0.875000
244	general_motors	69	59	0.855072
114	ua	124	106	0.854839
382	under_armour	47	40	0.851064

Figure 16: top 10 sentiment for companies seen over 25 times.

My vision for the final result of this project would be something like the table in figure 16. An actual application of these tools would likely look at daily changes in sentiment for various companies. This data could then be used a a single feature in a much larger algorithm for exploring the stock market.

One of my biggest takeaways from this project was the emphasis on quality of data. It is a quick lesson to learn that a model is only as good as its data. I know that I could spend a lot more time considering how I pre-process some of the data before it is input into the these tools. It has also become very clear that good training data is hard to come by. The only good way to get good training data for this would be to hand label a large amount of reddit posts with both identified companies as well as their sentiment.

Overall I am very happy with my results and all that I learned about NLP and working with these tools. I am excited to continue to learn about some of the more advanced tools in this space.

References

1. Jurafsky, Dan, and James H. Martin. Speech and Language Processing. Pearson Prentice Hall, 2014.

2. Tomas Mikolov and Quoc V. Le and Ilya Sutskever. Exploiting Similarities among Languages for Machine Translation, 2013.

3. Hu, Jie & Li, Shaobo & Yao, Yong & Yu, Liya & Guanci, Yang & Hu, Jianjun. (2018). Patent Keyword Extraction Algorithm Based on Distributed Representation for Patent Classification. Entropy. 20. 104. 10.3390/e20020104.

4. Charan Gowda, Anirudh, Akshay Pai, and Chaithanya kumar A, “Twitter and Reddit Sentimental analysis Dataset.” Kaggle, 2019, doi: 10.34740/KAGGLE/DS/429085.imd

5. Gabriel Preda, “Reddit WallStreetBets Posts.” Kaggle, 2021, https://www.kaggle.com/gpreda/reddit-wallstreetsbets-posts/metadata

6. Raphael Fontes, “Reddit - r/wallstreetbets”. Kaggle, 2021. https://www.kaggle.com/unanimad/reddit-rwallstreetbets

7. Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

8. Bird, Steven, Edward Loper and Ewan Klein (2009), *Natural Language Processing with Python*. O’Reilly Media Inc.

9. Semantics with Word2Vec | The Skip-Gram Model, Some Probability, and Softmax Activation. (2019, November 27). [Video]. YouTube. https://www.youtube.com/watch?v=pO-qz6KuvLV8

10. Forbes, 200(1), 38-42. Format: Meagan Andrews, A. A. (2021, Feb 08). What The Rise Of The Millennial Investor Means For A Sustainable World

11. Chung, Juliet (January 31, 2021). "Melvin Capital Lost 53% in January, Hurt by GameStop and Other Bets". The Wall Street Journal. Archived from the original on January 31, 2021. Retrieved January 31, 2021.

12. Rehurek, R. & Sojka, P., 2011. Gensim–python framework for vector space modelling. NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic, 3(2).