CSC 501 - Assignment 3

Joe Howie

December 21, 2020

Contents

List of Figures

List of Tables

Abstract

We were presented with a large data set of tweets from a variety of troll accounts on Twitter. These accounts were labelled by Twitter as "Right", "Left", "local", or "news" trolls. Our goal was to identify language bias between different trolling groups as a method of predicting the account types of new unseen tweets (see Theme). To enhance performance of our analysis, the tweets text was configured through a series of processing steps (see Cleaning). Then the tweets were vectorized using the TF–IDF model (see Vectorization). The prediction model, solely based on language, was found to correctly identify tweets account types (see Analysis). Finally, we discovered that regardless of account type, trolls repeat themselves in consecutive tweets, which inflates word frequency. We suggest that collapsing consecutively posted similar tweets would improvement to the prediction model (see Conclusion).

1 Introduction

1.1 Theme

Twitter claims more than three million tweets indicated certain accounts were engaging in "Trolling" on a particular group or genre of other non-trolling accounts (presumably). Twitter, in their infinite wisdom, handed us a data set of millions of tweets with many attributes, including one of particular interest: `account_type`. Twitter conveniently did not inform us as to how they assigned the labels for user's `account_type`. By analyzing the content of the tweets we can attempt to predict/reconstruct how Twitter assigned these `account_type` labels.

1.1.1 Questions

Question 1 *What are the bias' in word choice of the most related users for each of the different account types?*

Question 2 *How well can we predict account types from the highly related users for each account type found in question 1?*

Question 3 *How often are users from question 1 of the different account types tweeting similar content consecutively in short time ranges?*

1.1.2 Goal

Questions 1, 2, and 3 are all integral in attempting to unveil how Twitter assigned these labels. First we look at the similarity in the language used by each of the different account types. To compare the language we will construct TD–IDF vectors for each tweet based on a vocabulary. Once we have similar tweets for each of the different types, we can compare test tweets to these groupings and attempt to predict the account type from the data results in question 1. We then analyze the similar tweets in question 1 to see if there are similarities between consecutively posted tweets by a single user.

1.2 Procedure

The tweets will be analyzed by quantitatively converting all the tweets to TF–IDF vector representations based on the corpus of words. Since there are over one million tweets in english, we will have to process the text to reduce the complexity of the vocabulary. Then we can use the vector representation to determine similarity between tweets based on the magnitude of their dot product. Ideally, the vocabulary has approximately the same size as the number of documents in the corpus to help reduce complexity.

1.2.1 The Data

The tweet data came in 13 separate csv files, so first we combine these into one csv file. In the same script we reduce the column attributes to the five columns we care about for analysis. We kept columns: author, content, account_type, post_type, and publish_date; which hold: the user name, the tweet, Twitter's assigned label, if this was an original post or retweet or quote, and the date and time the tweet was posted. The next step was to isolate the original tweets and save those to a new file. The date and time were combine into a python datetime object for later analysis. Before applying any vector modelling to the tweets, first we remove and refine the text.

1.2.2 Cleaning

All of the steps described here were handled by the script `Clean.py`

Vocabulary The TF–IDF vector gets a position for each word in the vocabulary plus each bi-gram transition in the corpus that is allowed by the vocabulary. We will discuss the series of steps taken to reduce the size and complexity of the vocabulary while holding as much information as possible. First we preform a contraction on the corpus, which expands words like "you've" to "you have" and so on. Since we care about the language used in the tweet, we will strip away all punctuation, capitalization, emojis, and url's. These attributes do express information in the tweet such as emotion, volume, and even more with emojis; but we are interested in the language not the emotional expression. We also lose Twitter's signature hashtag (#) when stripping punctuation, but the presence of a hashtag can be inferred most of the time by lack of spaces.

Stop Words Next burden to tackle in the vocabulary involve removing stop words from the tweets. Stops words will not give us much information as they are more for proper grammar in english, with little to no flexabilty in language choice. For instance, all trolls will likely use the word "I", but that will not identify them into any of the different account types. Stop words are scanned for twice: first by the script `Clean.py`, then again while the vectors are constructed—it's built-in to the tool we used. Now words like "and", "the", "a" etc are scraped from the tweets, reducing the vocabulary further.

Lemmatization This is the process of converting english verbs to there base form. Hence words like "helping", "helpful", "helps" all lemmatize to "help". Now variants of the same verb will be compared in the dot product even if they were in different tenses in the tweets. Lemmatization really helps our analysis since we care about the language used, and this process reduces variants of the same verb to the comparable base word. Now even though english syntax dictates one tweet say "helps" and another say "helping", they mean the same thing ultimately, and now that is reflected in the quantitative analysis.

Numbers In tweets, numbers come in text and symbol form, which obviously adds complexity to the vocabulary. We made the choice to drop the symbol numbers and keep the texts instances. Numbers, like stop words, are generally used and are not indicative of language bias. However, text numbers were kept, otherwise too much information was lost. Often if a tweet contained a symbol number there was other text qualitatively quantifying the same information—ie "10 million..." or "46th anniversary..."—both of these example demonstrate why we can get away with dropping the symbol but not the text form of numbers.

Summary After all the cleaning was applied to the vocabulary, the size had drastically shrunk from being far larger than the number of tweets to nearly equal in size. Also, while much data was removed in the cleaning step we managed to hold on to the key language choice components, which directly relate to the three outlined questions.

1.2.3 Vectorization

Now that the corpus has been cleaned, we can transform the text data into a multi-dimensional vector representation.

TF-IDF Let m be the number of tweets selected for analysis. Let the corpus of documents $d_0, \ldots, d_{m-1}$ be the selected tweets. Let the vocabulary be the collection of words in the corpus that belong to at least two documents $d_i, d_j \ s.t. \ i \neq j$. Let t_i be the number of documents containing transition i. Formally,

$$IDF(i) = \log\left(\frac{N}{t_i}\right) \tag{1}$$

where the base of the logarithm is moot, because we care about the relative measure, not the absolute scale. So far we only have unique values for the transitions, and nothing for the documents. The TF in TF–IDF forms the basis for each document vector. TF stands for term frequency, the vector has a dimension for each transition—uni-gram and bi-gram—in the vocabulary, where the value is the frequency of that transition in the document. Formally,

$$TF(d_i) = \langle f_1^i, \ldots, f_n^i \rangle \tag{2}$$

where f_j^i is the frequency of transition j in document d_i, and n is the number of transitions with a mapping from word transitions (uni-gram and bi-gram) to vector positions. Then the product of each frequency with that transition's IDF value is the TF–IDF vector representation.

$$TFIDF(d_i) = \langle f_1^i IDF(1), \ldots, f_n^i IDF(n) \rangle \tag{3}$$

$$TFIDF(d_i) = \langle v_1^i, \ldots, v_n^i \rangle \tag{4}$$

Now for each document d_i we have a vector $TFIDF(d_i)$ of length n. Hence we have a $m \times n$ matrix containing each of the m document vectors as rows.

$$T = \begin{bmatrix} TFIDF(d_1) \\ \vdots \\ TFIDF(d_m) \end{bmatrix} = \begin{bmatrix} v_1^1 & \ldots & v_n^1 \\ \vdots & \ddots & \vdots \\ v_1^m & \ldots & v_n^m \end{bmatrix} \tag{5}$$

We now have a vector representation for each tweet in the corpus, where the vector positions all correspond to uni-gram and bi-gram transitions. Hence, by taking a dot product of any two tweets, we can infer their similarity to one another by the magnitude. The vectorizer library used normalized the vectors, meaning a large dot product is close to 1 (tweets are highly related in language) and a low dot product is close to zero (tweets are not that related). The bottleneck of the entire procedure is generating the matrix T. The calculation depends on the number of tweets, m, and the number of transitions, n, which is proportional to the vocabulary size. The next section will describe how matrices like T will be used to answer our three questions.

2 Analysis

2.1 Question 1

We want to find words that are common to users in the account types that twitter provided. About 2000 tweets where selected for each of the four types we analyzed: Right, Left, local, news. These type were specially chosen because of how integrated their content can be. Left and Right are diametric in politics and we expect some sharp contrast in language accordingly. Local, and news are more mundane in a sense, and thus we expect these to be similar to each other in language (because they are both news categories). However, politics does live at the centre of the news cycle, so perhaps there will be some more overlap than expected.

2.1.1 Method

We feed each of the four groups of tweets through the `analysis1.py` script. First the vocabulary is calculated to only include words that appear in more than one document, because otherwise the word will never contribute positively to a dot product with another tweet. The matrix T_1 is constructed where $m \approx n \approx 2000$, and we want to use T_1 to find the dot product of each tweet with each other tweet. Mathematically, we only need to multiply T_1 with transpose of T_1. Formally,

$$T_1T_1^T = \begin{bmatrix} v_1^1 & \dots & v_n^1 \\ \vdots & \ddots & \vdots \\ v_1^m & \dots & v_n^m \end{bmatrix} \cdot \begin{bmatrix} v_1^1 & \dots & v_1^m \\ \vdots & \ddots & \vdots \\ v_n^1 & \dots & v_n^m \end{bmatrix} \tag{6}$$

contracting the product of matrices with dimensions $m \times n$ and $n \times m$ we get $m \times m$ matrix, that is $T_1T_1^T = D_1$:

$$D_1 = \begin{bmatrix} \sum_{i=1}^n v_i^1 v_i^1 & \dots & \sum_{i=1}^n v_i^1 v_i^m \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^n v_i^m v_i^1 & \dots & \sum_{i=1}^n v_i^m v_i^m \end{bmatrix} \tag{7}$$

where each entry i, j in matrix D_1 is the dot product between tweets d_i and d_j, thus D_1 is a symmetric matrix. Next we find the dot products which had values greater than 0.5 and save those tweets to a new csv file. The saved csv file now contains only tweets that are similar to at least one other tweet in the corpus. These csv files will be used in the following two questions as well. The csv files were passed to `Visualize.py` to produce word clouds of the most frequent words in the file of related tweets.

2.1.2 Results

The selection of our four sub sets were taken from specific time frames summarized in table 1. The time frames were chosen semi-arbitrarily, we wanted each set to contain roughly the same number of tweets. We also wanted the tweets to be from roughly the same time period, so people are talking about a similar set of current events.

Type	From	to
Right	2017-6-10	2017-6-30
Left	2017-6-30	2017-9-30
local	2017-6-14	2017-6-20
news	2017-5-15	2017-11-15

Table 1: Epoch of tweets by account type

Each of the four account types we analyzed had similar dimensions, and all ran in under an hour. A summary of the matrix dimensions and the run times are provided in table 2. At small scales, the size of the matrix was proportional to the run time in seconds, provided the matrix was roughly square. That is, for matrix size 2000, the run time was about 2000 seconds.

Type	m	n	Run time (s)
Right	2259	2214	1979
Left	2151	2639	1910
local	2757	2519	3067
news	2079	1700	1861

Table 2: Matrix construction run times

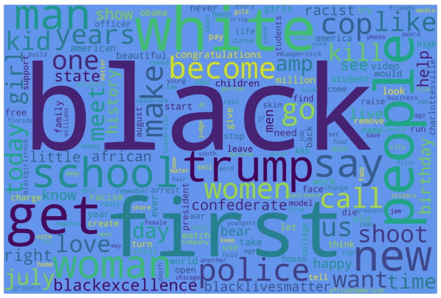

Figure 1: Most related words: Left

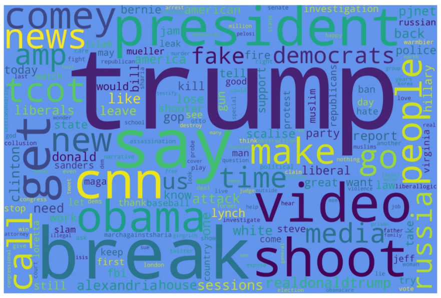

Figure 2: Most related words: Right

Figures 1, and 2 show the most frequently used words from the Left and Right trolls. Figures 3, and 4 show the most frequently used words from the local and news trolls. These were generated from the question 1 solution csv files produced from comparing the dot product of tweets. The size of the words indicate the relative frequency in the corpus. Here, the corpus for each account type are all the words from the tweets that had a dot product of 0.5 with at least one other tweet in the corpus. Formally, let C be the corpus of documents $C = \{d_i : For\ any\ d_j \in C,\ d_i \cdot d_j > 0.5\}$.

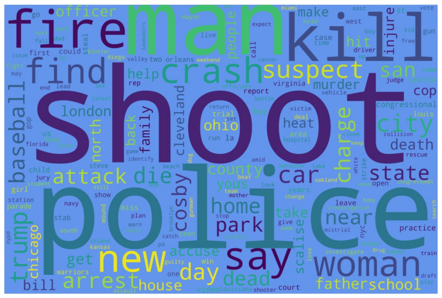

Figure 3: Most related words: local

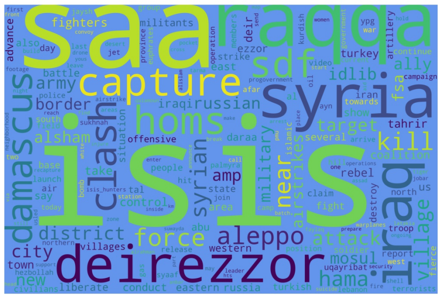

Figure 4: Most related words: news

2.1.3 Discussion

Clustering the high similarity tweets by the four selected account types, there are definitely sharp contrasts in words chosen by these accounts. At first glance of these four figures, each has some words that pop out more than other. Generally these words that pop out we can conceive of the relationship of these words, and messages one might use to "Troll" the four different target groups. Doing a quick `grep` on the original data set reveals that words like "white" and "black" are used in racially charged ways to incite a reaction from "the Left" (examples redacted due to inappropriate content). Similarly for "the Right", words like "trump", and "obama" are used to incite a tribal political reaction. In "the local" words like "shoot", "police", and "kill" are used in attention gripping fashion, that is a sensational headline (click bate content). Similarly with "the news" words like "isis", "syria", "iraq", "clash", "capture" are headlines geared towards global conflicts (bad news sells). The point is because these account types are using different language more frequently than others, we can attempt to use these words (and the bigram transitions) to predict if other tweets are of the same account type based on their vector dot product similarity.

2.2 Question 2

Now we want to try and use the csv files of related tweets produced in question 1 to predict the labels of tweets not yet analyzed. A test set of data was prepared with roughly equal amounts of tweets from each of the four types, and around the same time frame. The test data time frames and sizes are summarized in table 3.

Type	# tweets	from	to
Right	573	2017-7-2	2017-7-8
Left	505	2017-10-1	2017-11-25
local	557	2017-6-21	2017-6-22
news	556	2017-1-20	2017-2-25

Table 3: Epoch and number of test tweets

These tweets were combined into one file of just under 2200 tweets for testing.

2.2.1 Method

For each of the four account types, the script `analysis2.py` produced two matrices T_{train} and T_{test}. The vocabulary for T_{train} contains only words appearing in at least two tweets, like in question 1. After the matrix T_{train} is constructed, the matrix T_{test} was made with it's vocabulary equal to the vocabulary used for T_{train}. The vocabulary of T_{test} is the same as T_{train} because we are going to compare all the tweets in T_{test} with those in T_{train} via dot product. Since similarity to tweets in T_{train} is how we are going to predict the label, words that appear in the corpus for T_{test} but not in T_{train} will not contribute at all to the prediction. Additionally, from a matrix multiplication stand point, when the vocabulary of T_{train} and T_{test} are equal, we can compare all the tweets at once with a simple matrix multiplication like in question 1. Formally, let n be the number of transitions allowed by the vocabulary, let m be the number of training tweets, and let k be the number of tweets in the test set. Then,

$$T_{train} = \begin{bmatrix} v_1^1 & \dots & v_n^1 \\ \vdots & \ddots & \vdots \\ v_1^m & \dots & v_n^m \end{bmatrix} \tag{8}$$

$$T_{test} = \begin{bmatrix} u_1^1 & \dots & u_n^1 \\ \vdots & \ddots & \vdots \\ u_1^k & \dots & u_n^k \end{bmatrix} \tag{9}$$

where T_{train} is a $m \times n$ and T_{test} is a $k \times n$. We want the dot products $d_i \cdot d'_j$ where d'_j are training tweets and d_i are the test tweets. This can be accomplished by multiplying T_{test} by T^T_{train} to produce D_2:

$$T_{test}T^T_{train} = \begin{bmatrix} u_1^1 & \dots & u_n^1 \\ \vdots & \ddots & \vdots \\ u_1^k & \dots & u_n^k \end{bmatrix} \cdot \begin{bmatrix} v_1^1 & \dots & v_1^m \\ \vdots & \ddots & \vdots \\ v_n^1 & \dots & v_n^m \end{bmatrix} \tag{10}$$

$$D_2 = \begin{bmatrix} \sum_{i=1}^n u_i^1 v_i^1 & \dots & \sum_{i=1}^n u_i^1 v_i^m \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^n u_i^k v_i^1 & \dots & \sum_{i=1}^n u_i^k v_i^m \end{bmatrix} \tag{11}$$

D_2 is a $k \times m$ matrix where the entry i, j is the dot product of tweet d_i from the test set with tweet d'_j from the training set. Unlike D_1, D_2 has no symmetry; D_2 does have the property that each row vector holds the dot product of one tweet in the test set with all the tweets in the training set. To predict if a tweet in the test set is from the same account type as the training set tweets, we sum all the dot products in each row and make a selection cut on that value to determine whether the tweet is of this account type or not. Hence, we wish to collapse D_2 to a column vector where each entry i is the sum of all the dot products of d_i with all the tweets in the training set. That is, we multiply D_2 be a column vector of length m containing ones to produce D_{sum}.

$$D_2 \cdot \vec{1} = \begin{bmatrix} \sum_{i=1}^n u_i^1 v_i^1 & \dots & \sum_{i=1}^n u_i^1 v_i^m \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^n u_i^k v_i^1 & \dots & \sum_{i=1}^n u_i^k v_i^m \end{bmatrix} \cdot \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \tag{12}$$

$$D_{sum} = \begin{bmatrix} \sum_{j=1}^m \sum_{i=1}^n u_i^1 v_i^j \\ \vdots \\ \sum_{j=1}^m \sum_{i=1}^n u_i^k v_i^j \end{bmatrix} \tag{13}$$

Now we have a vector with numbers measuring the similarity of test tweets to each of the training tweets. By applying a threshold cut we can predict which of the k tweets belong in this account type, and measure the error using confusion matrix statistics.

2.2.2 Results

After running `analysis2.py` on all four training sets—that is all four of the resulting csv files from question 1 for Left, Right, local, and news—we were left with four D_{sum} vectors, one for each account type. These were passed to `results.py` where the threshold cuts were applied to the D_{sum} vectors to determine which tweets belonged to the account type. Once the cuts were applied, the true labels were used to get the confusion matrix statistics, the results are summarized in table 4.

	Right	Left	local	news
Threshold	20	15	20	15
Accuracy	0.772	0.768	0.735	0.901
Precision	0.463	0.497	0.468	0.753
Recall	0.394	0.432	0.303	0.908
F-Measure	0.426	0.462	0.368	0.823

Table 4: Confusion matrix results of predictions

Again the run time was constrained by the construction of the T matrices, which still took under an hour to produce for each training account type and the test set. Since each run constructed two different T matrices, each run of `analysis2.py` took under 2 hours—about 4000 to 5000 seconds each.

2.2.3 Discussion

From table 4, we immediately see that using the language bias identified in question 1—the saved csv files—does do a good job of predicting Twitter's assigned account types to previously unseen tweets. As there are four distinct account types, a random guess would produce an expected accuracy of 0.25. With the exception of news [1] , all of the account types had a prediction accuracy of about 0.75—far better than random. However, precision and recall were not great, an indication that Twitter likely used more than just language bias when determining these account labels. Perhaps they even used a completely different metric to allocate these grouping, and the language bias discovered was a byproduct.

[1] Note, news likely did much better, in part, because the time span of the data was much larger than the other account types.

2.3 Question 3

So far our questions only looked at how similar tweets were to each other and used the vector representation to compare tweets with dot products. Now we look at the tweets from just an individual user in each account type, and analyze the similarity between consecutively posted tweets along with the time span in which they occurred. That is, we compare the consecutive tweets from an individual account and if they meet a threshold in their dot product we record the tweets and measure the difference in time. The reason behind this is to uncover other measures potentially used by Twitter to label accounts. It could be the case that twitter looked for user who spam tweet the same or similar content quickly—on the order of minutes to tens of minutes.

2.3.1 Method

The result csv files from question 1 were passed to `analysis3.py` where one user was selected from each of the four files for the analysis. The vocabulary only included words present in at least two tweets in the corpus of that user's tweets. The matrix T_3 was constructed for each of the four account types, and the dot products were calculated same as in question 1. The produced matrix of dot products D_3 was different than D_1 because it used only tweets of one user, and these tweets came from the answer to question 1—hence they were similar to other tweets in the file, possibly other users tweets. We checked the resulting dot products of consecutive tweets and regarded the content as similar if the dot product was at least 0.25. The output of the script was an arrays of points, the first was the time difference for the cluster of similar consecutive tweets, and the second was the number of similar consecutive tweets in the cluster. These point pair arrays were sent to `timePlot.py` where the number of consecutive pairs in each cluster were counted, and a histogram was produced showing the amount of similar content posted in different time spans—ie if one user posted 10 similar tweets in 30 minutes, that is 9 consecutive pairs in 30 minutes, this corresponds to 9 occurrences in the 30 minute bin on the histogram.

2.3.2 Results

Once again, the hangup in the process was constructing the matrix T. This time because only one user from the question 1 results was analyzed here, the number of tweets varied in each account type. The vocabulary size, document size, and user name for each account type are summarized in table 5 along with the run times. The histogram shows what time ranges different account types tweeted similar tweets. Generally speaking, from the histogram we note that users often tweet similar tweets in short time spans (0 - 40 minutes).

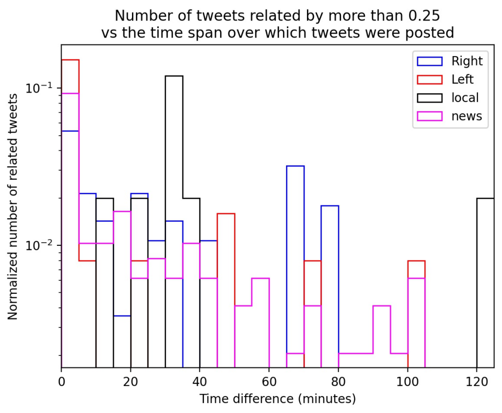

Figure 5: Frequency Histogram of postings in time ranges

Type	User	# tweets	Vocabulary	Run time (s)
Right	PAMELA_MOORE13	424	698	388
Left	CRYSTAL1JOHNSON	758	1155	574
local	ONLINECLEVELAND	211	285	177
news	TODAYINSYRIA	1801	1351	1174

Table 5: User selection, matrix dimensions, and construction run times

2.3.3 Discussion

From the histogram we see that users often tweet back to back effectively the same message when trolling. This result makes intuitive sense, if you want to rial up as many people as possible, tweeting the same or similar thing many times increases the likelihood that more targeted users see and share the tweet. Moreover, different account type display different spaced patterns. However, these were only from one user in each account type, so the generalization of these patterns to the larger set of users in the same account type is limited. Clearly the frequency of some transitions in question 1 were inflated due to repeating similar tweets. The inflation caries over into the prediction model for question 2 since it was based on results produced in question 1.

3 Conclusion

We were handed 3 million tweets from alleged trolls of different account types: right, left, local, and news. We attempted to devise a strategy for re-deriving the account type labels provided by Twitter. We found that these four groups have distinct words and two word transitions that are commonly shared between related tweets (see question 1). This sparked the idea that perhaps we would be able to reconstruct the account type labels for unseen tweets by comparing them to these recently found related tweets for each group. With decently high accuracy, we were able to re-derive account type labels for most unseen tweets just based on the document vector representations (see question 2). Finally, users from all account types were found to repeat themselves in consecutive tweets—some more frequently than others—which has the property of elevating the frequency of their identified language bias (see question 3). Ultimately we learned two things from this analysis. Firstly, when trolling, language plays a keys role in targeting a response, and is therefore a good predictor of other trolling tweets of that account type. Secondly, trolls repeat themselves, making consecutive posting of similar tweets a good indicator of trolling behaviour. The repeating nature of trolls tells us that perhaps to improve our predictions, we should remove/combine highly related consecutive tweets for the same user—reducing the artificial inflation of the repeated language—to improve/refine the prediction model. It is likely that Twitter used many different metrics—possible not even in this data set—to produce these account type labels. Therefore, the fact that we were able to reconstruct reasonably well their results and identify a factor for improving the method of prediction just from TF–IDF document embeddings really speaks to the power of this form of text based vector analysis.

4 Acknowledgements

Special thanks to the numpy , pandas , and matplotlib docs, without their help this code would have been horribly inefficient. Thank you to Dr. Sean Chester for all the help and guidance on this assignment. And of course thanks to the collective knowledge of people on troubleshooting forums such as: Stack Exchange , Geeks for Geeks , and many more. Note sci-kit learn documentation was referenced as well.

5 Appendix: Code

See full code on my GitHub page. Please check it out!