Comparing Classifiers of Sentiment for Short Text

Mitch Kinney

December 12, 2017

Introduction

For my end of the semester project I will explore a new model to do sentiment analysis on short texts using methods developed in the class STAT 8931. During class we were introduced to Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA) which are parametric methods that take advantage of the Normal likelihood to separate data. We studied these methods in the context of different precision matrix estimates we could supply the classifier. I will create the numerical representations of the tweets using a generator I've been exploring in my own research called doc2vec [4] which is able to convert documents of words to numerical vectors. The authors claim it is able to capture the context of a sentence which is helpful in tasks such as sentiment analysis. The dataset I will be using is Stanford's Sentiment140 dataset [2] which is a collection of 1.6 million tweets that are pre-labeled with a positive or negative sentiment. My main goal is to compare these classifiers to see what level of tuning parameters are most beneficial to a correct classification rate and help understand the relationship between the data, the generator and the classifiers.

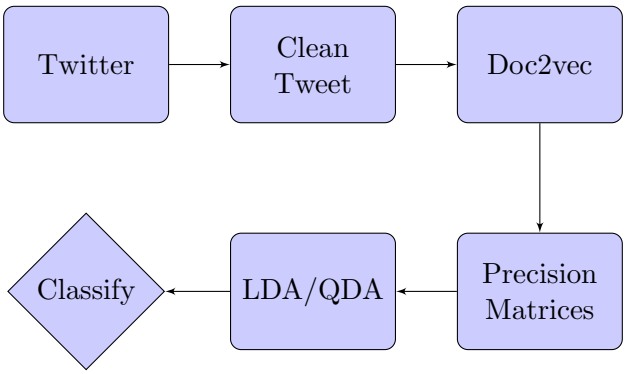

Figure 1: Flow diagram of project

Previous Work

In the last 20 years or so there has been a lot of work on analyzing text data thanks to the increased computing power we have access to and because of the interest to train mathematical models that can capture information about text that is easily discernible by humans. Sentiment in short texts is one of these problems that researchers have attempted to tackle. Many of the methods utilize a combination of techniques that can be broken down into two parts: first generate a numerical representation of the text data and second use existing model machinery to build a classifier. The generator of the numerical representation will usually be an unsupervised method and then the labels will be used in training and testing the classifier. This allows for a lot of speculation on if the success of these combinations rely more heavily on the generator or the classifier or if there is some dependence on the combination itself (if methods compliment each other for example). Since I am doing an analysis on only one dataset I am unable to explore this open question in my paper so I will focus on comparing the success of the classifier in this specific situation. There are a multitude of options in how to combine generators and classifiers when doing sentiment analysis which has been the subject of many other papers.

One example of a combination comes from Dos Santos and Gatti [9] in which they did sentiment analysis on the same tweets dataset that I will use. They utilized word2vec to generate numeric representations of the words in each tweet and then used a Convolutional Neural Network (CNN) to classify. Word2vec [5] is a algorithm designed by researchers at Google DeepMind that is able to capture the semantic relationship between words which I will go into more depth later on. The author's method was able to achieve state of the art performance (at the time) classifying correctly at an $\sim$85% accuracy rate. In their CNN the authors took advantage of moving context windows and different character capitalization to capture the sentiment of the tweets.

Another example of a proposed combination was detailed in Go et al. [2] where they suggested to use a bag-of-words (BOW) approach in combination with a Naive Bayes estimator. This paper was published in connection to the Sentiment140 dataset I will be using too. BOW is an old and simple approach to generate a numerical representation of text by creating a matrix of ones and zeros with the number of columns equal to the number of unique words (or number of unique words of interest) in all sentences and number of rows equal to the number of sentences. There will be a one in the i^{th}, j^{th} cell if word j is in sentence i. Otherwise there will be a zero. The choice of a classifier is a multinomial Naive Bayes model which trains itself based on if certain words are in the tweets. With this method they were able to achieve a classification accuracy of ∼81%. This showed that simple models can be effective in sentiment analysis.

Doc2vec Overview

The generator that I chose to create a numerical representation of the text is called doc2vec which is an algorithm designed to capture the context of a document. According to the authors Le and Mikolov [4] doc2vec is specifically designed to predict the next word in a document given the surrounding words which is why it is able to build a numerical representation that provides context. Doc2vec was built off of an earlier designed algorithm by Mikolov et al. [5] called word2vec which was able to accomplish a similar task for words. In the paper for word2vec [5] the authors claim that the algorithm can capture the semantic relationships between words implying that words used in similar situations will have similar numerical representations. I will talk about both methods and how they can offer an advantage in sentiment analysis tasks.

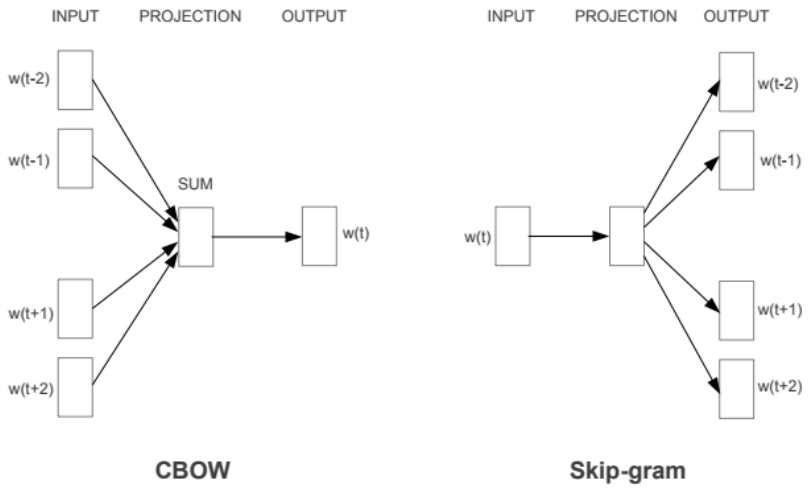

Figure 2: The Continuous Bag of Words and Skip-gram networks used in word2vec [5]

Word2vec

Word2vec is an algorithm that has gained a lot of attention since it's creation in 2013 because of it's unique ability to coerce words into numerical vectors that mimic the relationships found in language. It was built at Google DeepMind which is today mainly known for it's AI programs that can beat expert human players in games like chess and go. Word2vec uses a recurrent neural network to approach this natural language processing task. There are two different network architectures that can accomplish the same goal of creating vectors. The first one takes in a single word and attempts to predict the words surrounding it called Skip-gram (Figure 2). The other one called Continuous Bag of Words (Figure 2) takes in words in a window around a target word and attempts to predict the target word.

In mathematical notation they are both trying to maximize

$$\log P(w_T|W_c)$$

Where w_t is the current target word and W_c is the context words being used. This is done for random words in the documents provided. Both models use stochastic gradient descent to train the parameters in the model. Since the cost function in both networks is whether the prediction of the target word is correct the actual numeric vectors that act as the representations of the words have to be extracted from the middle of the network and are trained parameters rather than output. As a result of this the vectors representing the words have interesting properties when doing vector math on them. For instance the authors use the example that the vector for the word "king" minus the vector for the word "man" plus the vector for the word "woman" will equal a vector close in cosine distance to the vector for the word "queen". Interestingly the authors are only able to give a high-level explanation of why this method works rather than a technical and mathematical one. This has led to other papers such as Goldberg [3] and Rong [6] to attempt to explain what is going on but neither paper is able to really give a full explanation. Doc2vec shares similar characteristics but is able to extend the idea to phrases and sentences.

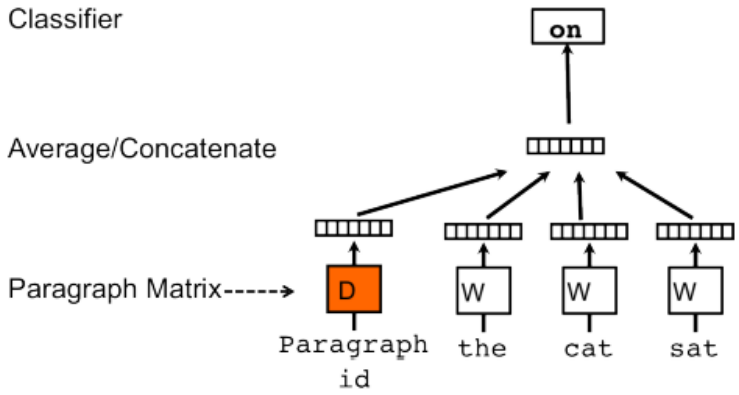

Figure 3: Network architecture of doc2vec [4]

Doc2vec

Created about a year later with Mikolov as one the authors on the paper, doc2vec incorporated a similar strategy as word2vec to capture the context of documents. Doc2vec is also built with a neural network that focuses on predicting words to build the numerical representations of documents. The main difference lies in the addition of a paragraph vector as well as word vectors to predict words. While word2vec only uses words in a window around a target word, doc2vec uses another vector to try to capture the overall context of the document that uses information from word predictions that happened in other parts of the same document (Figure 3). Similar to word2vec there are two different network architectures that can be used with doc2vec. The first one is the parallel to Skip-gram called Distributed Memory (DM) that attempts to predict a target word based on preceding words and the paragraph vector. The second method is the Bag of Words (BOW) model and predicts the target word based on an unordered collection of context words surrounding the target word and the paragraph vector. Both methods are trained the same way trying to maximize

$$\log P(w_T|W_c, d_i)$$

Where w_T is the target word, W_c are the context words, and d_i is now the paragraph vector for document i. The authors stress that each of these paragraph vectors will differ between documents but word vectors will be shared across documents. Note that while this method also creates numerical representation of the words only the paragraph vectors are of interest. Something that doc2vec must do differently is generate paragraph vectors for documents unseen in training. The authors do this by combining a randomly generated vector representation of the new document with the matrix of already

trained paragraph vectors and run the neural network again until convergence criteria is met holding the word vectors fixed. So the word vectors do not get the benefit of the context the new document provides. There is no high level interpretation of the paragraph vectors as there is with word2vec since defining the semantic relationship between documents is much more difficult. Results from the paper showed the method outperforming other current methods in different areas of interest as well as models based on bag of words. It is the hope that numerically representing relationships will be a step forward in all non-numeric data. For this project though I will be focusing on the combination of using doc2vec generated vectors with parametric classifiers such as LDA and QDA and using different estimates of precision matrices.

Discriminant Analysis

A common place to start when doing classification for groups is by doing discriminant analysis. The two types of discriminant analysis are Linear (LDA) and Quadratic (QDA). They are based on using a probabilistic approach to draw boundaries around known data points in order to classify new ones. This type of analysis requires a lot of data spread across all the groups since the estimators for each are derived independently based on the group. The main assumption of the method is that all the data comes from a normal distribution where the different groups will have different means. The difference between LDA and QDA is in LDA the covariance matrices are assumed the same for all the groups while in QDA the covariance matrices are assumed different for all the groups. Using the notes from Adam Rothman [7] to put things into mathematical terms suppose the data is $(X_i, y_i),\ i = 1, ..., n$ where $y_i \in \{1, 2, ..., C\}$ and $P(y_i = c) = \pi_c$. Then

$$X_i | y_i = c \ \sim \ N(\mu_c, \Sigma_c)$$

When we are using the LDA model we assume that $\Sigma_1 = \Sigma_2 = ... = \Sigma_C$ whereas in the QDA we assume they are all different. To estimate these parameters we use their MLE's. Solving for these involves using a Lagrangian (from our first STAT 8931 homework). In the end under the QDA model the MLE's will be group dependent on N_c being the number of observations in group c and are

$$\begin{aligned}
\hat{\pi}_c &= \frac{N_c}{n} \\
\hat{\mu}_c &= \frac{1}{N_c} \sum_{i=1}^{n} X_i \mathbf{1}(y_i = c) \\
\hat{\Sigma}_c &= \frac{1}{N_c} \sum_{i=1}^{n} (X_i - \hat{\mu}_c)(X_i - \hat{\mu}_c)^T \mathbf{1}(y_i = c)
\end{aligned}$$

And in the LDA model the only change is

$$\hat{\Sigma} = \frac{1}{n} \sum_{k=1}^{C} \sum_{i=1}^{n} (X_i - \hat{\mu}_k)(X_i - \hat{\mu}_k)^T \mathbf{1}(y_i = c)$$

After training, in order to classify a new point X_{new} we use Bayes rule and get the probability of the point belonging to group c to be

$$\begin{aligned}
P(y_{new} = c | X_{new}) &= \frac{P(X_{new} | y_{new} = c) P(y_{new} = c)}{\sum_{k=1}^{C} P(X_{new} | y_{new} = k) P(y_{new} = k)} \\
&= \frac{P(X_{new} | y_{new} = c) \pi_c}{\sum_{k=1}^{C} P(X_{new} | y_{new} = k) \pi_k}
\end{aligned}$$

And choose the argument c that maximizes $P(y_{new} = c | X_{new})$ to assign to y_{new}. Since we assume our data comes from a normal distribution we will use the normal density to calculate these probabilities. Assuming $X | y = c \sim N(\mu_c, \Sigma_c)$

$$P(X | y = c) = (2\pi)^{-\frac{p}{2}} |\Sigma_c^{-1}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2} (X - \mu_c)^T \Sigma_c^{-1} (X - \mu_c) \right)$$

Then to compute estimates of the probability we can substitute in the MLE estimates of $\hat{\pi}_c$, $\hat{\mu}_c$, $\hat{\Sigma}_c$. One major hurdle to justify when using this classifier is that the data is assumed coming from a normal distribution. Since the data I will be supplying to the classifier is generated from an algorithm a normality assumption would be unreasonable. Luckily justification for exploring LDA/QDA as a possible classifier comes from Friedman et al. [1] which says

> ...a reason is that the data can only support simple decision boundaries such as linear or quadratic, and the estimates provided via the Gaussian models are stable.

My interpretation of this is that since these discriminant analysis methods have shown to be basically drawing boundaries around the trained data, any sort of data that can cluster should have good results. The key to the LDA/QDA though are how these boundaries are drawn and a big influence is the precision matrix.

The precision matrices that I will compare are the inverted MLE covariance matrix, the ridge penalized precision matrix, and the inverse of the diagonal elements of the MLE covariance matrix. I have already detailed the MLE estimate of the covariance matrix and since I am in a situation where $n > p$, I am able to use its inverse which I will call $\hat{\Omega}^M = \hat{\Sigma}^{-1}$. The diagonal inverse matrix can be represented as $\hat{\Omega}^D_{i,i} = \frac{1}{\hat{\Sigma}_{i,i}}$ and for $i \neq j$, $\hat{\Omega}^D_{i,j} = 0$. Finally, the ridge penalized precision matrix is the solution to

$$\hat{\Omega}^R = \underset{\Omega \in \boldsymbol{S}^p_+}{\arg\min} \left\{ \text{tr}(\hat{\Sigma}\Omega) - \log|\Omega| + \frac{\lambda}{2}||\Omega||^2_F \right\}$$

To get a closed form solution see Adam Rothman's notes [7]. We first need that the singular value decomposition of $\hat{\Sigma} = VQV^T$. Then plugging into the solution we get

$$\hat{\Omega}^R = \frac{1}{2\lambda} V\{-Q + (Q^2 + 4\lambda I)^{\frac{1}{2}}\}V^T$$

Note that λ is the penalizing term that will be chosen based on cross validation. There is no clear reason in the case of the doc2vec vectors that shrinkage of the precision matrix would be helpful since the parameters of the data is user controlled. Therefore I would expect that a very small λ value would be chosen for $\hat{\Omega}^R$. Also that $\hat{\Omega}^D$ would not be a good representation of the true precision matrix for the same reason that shrinkage is not expected to be needed. However we have seen that shrinkage can help in estimating precision matrices even when shrinkage is not required. Since the main objective of this study is classification, having a good estimate of the true precision matrix will be extremely helpful.

Results

Dataset

The dataset I am using to test the combination of doc2vec as a generator and LDA/QDA as a classifier is the Sentiment140 [2] dataset originally created by students at Stanford. This dataset is comprised of 1.6 million tweets that have a positive, negative and neutral sentiment attached to them. The dataset was collected by searching key words in combination with smiley faces ":)" and frowny faces ":(". The tweets with smiles were labeled positive and the tweets with frowns were labeled negative. It is not clear from their website how the neutral sentiment was assigned, but in this project I will only use positive and negative tweets. The tweets were automatically assigned sentiment rather than manually assigned sentiment which made the process extremely fast. In my experiments I chose to use a maximum of 200,000 tweets so I am not touching a majority of the data. The state of the art classification rate for this data is about $\sim$85%.

Figure 4: Example positive tweet

Coding

This project is an off-shoot of my current research so I will have code in Python and in R. The code in Python will be to create the actual doc2vec vectors from the tweets and then I will use Adam Rothman's code in R to do training and testing with the LDA/QDA models using the different precision matrices. The main library that I will be using in Python is gensim which is a Python implementation of the papers [4], [5] that are not done by the authors. The actual function of doc2vec in gensim has a lot of tuning parameters that are required to get the best classification rate. Some of these tuning parameters for example are context window size, length of the outputted vectors, etc. For this project I will use a lot of the parameter choices that I have found successful in the past and only adjust length of the outputted vectors. I believe these tuning parameters will have the a smaller impact on classification than the number of samples seen since- especially in text data- the more samples the better. Other parameters that papers I have read recently also suggest that the adjusting the learning rate is important. To get a classification rate I will use five fold cross validation. I was not able to find a reasonable way to run everything through one console so I am using an intermediate file to store the doc2vec vectors and then have the R scripts pull from that folder. To run everything took approximately 30 hours of computing time.

Comparisons

I chose to make many types of comparisons to see how I could extend this work to get the best overall model. I will compare different lengths of doc2vec vectors, different methods of doc2vec, different number of samples supplied, the different precision matrix estimates and the difference between the LDA and QDA model. When looking at the results it was clear that using the precision estimate of $\hat{\Omega}^D$ did extremely poor. For this reason I have chosen not to include results using it. First I will present the table of percent of accurate predictions when using the LDA model with both doc2vec methods DM and BOW. In the columns I will have the number of samples and in the rows I will have the length of the outputted doc2vec vectors. For each cell I will have two classification rates representing using the precision matrices $\hat{\Omega}^M$ and $\hat{\Omega}^R$ respectively.

DM

Length \Samples	25,000	50,000	100,000
250	(0.709, 0.710)	(0.722, 0.723)	(0.728, 0.729)
500	(0.710, 0.720)	(0.726, 0.730)	(0.732, 0.735)
1000	(0.701, 0.719)	(0.721, 0.730)	(0.731, 0.737)

LDA

BOW

Length\Samples	25,000	50,000	100,000
250	(0.511, 0.512)	(0.506, 0.516)	(0.515, 0.517)
500	(0.509, 0.518)	(0.507, 0.515)	(0.511, 0.518)
1000	(0.501, 0.521)	(0.505, 0.518)	(0.508, 0.519)

Before I report the results for the QDA I'll make a few observations. First is clearly the DM method of using doc2vec is superior to BOW. This was expected as DM takes into account context and order whereas BOW only cares about whether certain words are present. Also the training dataset was split 50\50 so using LDA with BOW isn't much better on average than assigning everything to a single class. The next observation is that changing the length of the outputted doc2vec vectors does not seem to make much of a difference. Since computing the precision matrices is directly affected by the length of these vectors it is good thing to know that performance is not improved by injecting more parameters. This is an interesting result for my own research too as it seems there is a cut off for length of the vector where information seems to be capped. The next thing to notice is that shrinkage on the whole was better than using the MLE but only marginally. Using the ridge penalized precision estimate across the board did better than the MLE but in some cases only in the thousandth decimal place, so it is unclear if it is worth the extra computing power to use shrinkage. The biggest contributor to a better classification rate was number of samples. For DM especially each additional jump in samples used contributed to an almost 1% increase in the classification rate. I intend to follow up on this and increase the number of samples used in building the doc2vec vectors while keeping the length lower. Now I will show the results when using the QDA model. Since BOW had a similar performance I will only show the table for DM.

QDA

	DM		
Length \Samples	25,000	50,000	100,000
250	(0.676, 0.676)	(0.695, 0.694)	(0.686, 0.686)
500	(0.675, 0.677)	(0.696, 0.694)	(0.694, 0.691)
1000	(0.669, 0.677)	(0.693, 0.693)	(0.699, 0.692)

The first thing to see is that the LDA model did much better than the QDA model on the whole. I should also note that both models were run using the same indices for the folds so comparison is fair. The QDA model is less clear about what helps most with the classification rate as increasing the number of samples did not always help. Neither was length of doc2vec vectors or the difference between using the MLE or ridge penalized precision estimates uniform in doing better or worse.

Overall I would conclude that these results make sense for the most part with a few surprises and some things to follow up on. The two results that make the most sense is that using DM and more samples will result in better classification rate. Since DM takes into account words in an order and not just whether they are present or not it should theoretically do better in a sentiment analysis task. And increasing the number of samples will make every statistical analysis work better. Some things that are interesting to note is that when using the LDA model shrinkage does help even if only slightly and that increasing the length of the doc2vec vectors does not have much impact on the classification rate. It is worth noting that when selecting the lambda in cross validation for the ridge penalized precision matrix estimate the lambda chosen was always the lowest on the boundary. This would suggest that shrinkage is not necessary but the fact that it was able to do marginally better suggests that shrinkage possibly helped guard against overfitting. The length of the vector having no affect on classification rate implies that the information from the doc2vec vectors remains basically constant as the length increases. To save on computation time I can reduce the length and still get a good classification rate. As I have discussed previously there is no really good mathematical explanation for what these vectors are doing or capturing so it is good to know that there is some sort of upper bound on information. The next thing to explore is if a length of 250 is the lowest I can go and still get the same classification rate and also if increasing the number of samples even more continues to raise the classification rate.

Further Work

In addition to testing the limits of decreasing the vector length and increasing the samples I would also like to attempt to use another precision matrix estimate with glasso, and practice coding the project in Julia. First I will address my attempt to use glasso. While this kind of precision matrix does not necessarily beg for shrinkage the fact that the ridge penalized estimate showed some promise made me want to expand to the lasso penalty. Unfortunately the glasso estimator in its natural state is not friendly to precision matrices with a lot of features. As was suggested by a peer a speed up is to do a one step estimator of the beta coefficients within the inner loop instead of waiting for convergence. I was unable to find the exact references that were used but I attempted the suggestion to see if I could have any success. To be more specific using Adam Rothman's notes [8] the glasso precision matrix estimate $\hat{\Omega}^G$ solves

$$\hat{\Omega}^G = \underset{\Omega \in \boldsymbol{S}_+^p}{\arg\min} \left\{ \text{tr}(S\Omega) - \log|\Omega| + \lambda \sum_{j \neq k} |\Omega_{j,k}| \right\}$$

This problem reduces to one of solving for the lasso penalized coefficients as shown in the notes. Namely there is a step involved in which at each k^{th} iterate and for every $j = 1, ..., p$ one needs to solve

$$\beta^{(k,j)} = \underset{\beta \in \mathbb{R}^{p-1}}{\arg\min} \left(\frac{1}{2} \beta^T \Sigma_{-j,-j}^{(k)} \beta - \beta^T S_{-j,j} + \lambda \sum_{m=1}^{p-1} |\beta_m| \right)$$

Solving this involves using the coordinate descent method so an individualized update needs to be applied for every parameter in the β coefficient. In my original study my minimum vector length was 250. This makes the updates very costly so I was hoping that by not waiting for the β convergence and only taking one step things would be sped up. After letting it run for a couple hours though it did not converge so I decided to also limit the number of outer iterations and stop the algorithm before it reached convergence for the precision matrix. This is definitely ill advised and only done in this situation to get an idea of if this is a route possibly worth following through on in the future. After the number of iterations were up I got a classification rate of 0.701. This is a pretty good result for terminating the algorithm early so it is possible that waiting until convergence could have better results.

Next I tried extending my grid for number of samples used and length of the doc2vec vectors. I used the LDA model to classify and the doc2vec method DM still as that combination proved best. I will still compare the MLE and ridge penalized precision estimates. The tables and graphs of the new points are below.

Length \Samples	100,000	200,000
150	(0.712, 0.714)	(0.714, 0.715)
200	(0.729, 0.728)	(0.727, 0.729)
250	(0.728, 0.729)	(0.730, 0.731)

From what I can see I can go down to a doc2vec length of 200 and still get comparable results but increasing the number of samples past 100,000 does not seem to help out much.

This project is a combination of ideas discussed in my STAT 8931 class and my own research. Therefore a lot of the code was already written either by myself or by Adam Rothamn. If given more time I would have liked to try out implementing the code for finding the estimates of the precision matrices and for classifying using LDA/QDA in Julia. I've only heard about Julia through word of mouth and by doing a brief glance at their website but I believe that it is an exciting new language that has a lot of potential. I would have liked to do some timing experiments to compare the run time between Julia and R to see if Julia's claim of speed up is true.

Conclusion

In this report I have outlined a new combination of generator and classifier to do sentiment analysis of short texts and tested parameters to determine what works best when classifying a dataset of positive and negative tweets. In the literature there are many examples of an unsupervised method of generating a numeric representation being paired with a model to classify the data with the additional information of labels. I chose to use a new algorithm to translate documents into vectors called doc2vec that claims to be able to capture the context of sentences. Then as a classifier I used LDA/QDA with different estimates of the precision matrices. The two main precision matrices were the inverse of the MLE covariance matrix and the ridge penalized precision matrix. Among the different parameters I tested were number of samples supplied, length of the vector outputted by doc2vec, and whether doc2vec should use DM, a context based approach, or BOW, a presence based approach. In the end I found results that were not too surprising and was able to follow up on them to explore how much I could push things. I found that increasing the number of tweets used in building the doc2vec vectors was the biggest help in increasing the correct classification rate, using the shrinkage precision matrix when paired with the LDA model offered some improvement, the length of the doc2vec vectors can be efficiently minimized without losing information and using the context based method of doc2vec, DM, is superior to BOW. This report has been both enlightening and great practice for a large scale experiment. Thank you!

Bibliography

[1] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning.* New York: Springer series in statistics, 2003.

[2] Alec Go et al. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 2009.

[3] Yoav Goldber. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. 2014.

[4] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014.

[5] Tomas Mikolov et al. Efficient estimation of word representations in vector spac. 2013.

[6] Xin Rong. word2vec parameter learning explained. 2014.

[7] Adam Rothman. Stat 8931 notes on ridge-penalized gaussian likelihood precision matrix estimation. 2017.

[8] Adam Rothman. Stat 8931 notes on an algorithm to compute the lasso-penalized gaussian likelihood precision matrix estimator. 2017.

[9] Cicero Nogueira Dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts". *COLING*, 2014.