STATS 202B Final Report
Dimensionality Reduction in Binary Text Classification Using Regularized Logistic Regression

Nicklaus Kim

March 19 2022

1 Abstract

One of the biggest problems taxing modern data analysis and modeling is that of computational cost and efficiency. This is especially true when working with sufficiently large, high-dimensional data matrices, a ubiquitous occurrence in the areas of text analysis and natural language processing. We investigate the practical benefits that are gained from considering dimensionality reduction techniques in the context of a binary text classification problem. To assess the impacts such a matrix approximation may have, we look at using regularized logistic regression to serve as our model so that we may compare predictive accuracy across models using full, high-dimensional data versus those using the reduced data.

2 Introduction

In today's data-plentiful world, dealing with the copious amounts of natural language data being continuously created has become a major point of focus in computer science and statistics, both in industry and in academic circles. As an increasing amount of raw text has become available from the World Wide Web, learning to handle such forms of data has become a necessity in modern times. Research into natural language processing (NLP) has become a crucial part of machine learning academia, information technology, marketing, and so many other fields over the past few decades.

This paper chooses to focus on one of the core subfields of NLP, authorship attribution. Currently, authorship attribution is a mostly academic problem, with the goal of correctly identifying the author of a given manuscript by studying its aspects such as word usage

(lexicon), style, sentiment, etc. One fascinating hallmark example of authorship attribution in practice is Frederick Mosteller and David L. Wallace's use of Bayesian methods to designate which of three Founding Fathers — Alexander Hamilton, James Madison, or John Jay — wrote each of the 85 *Federalist Papers* [1]. Another that is more applicable to everyday life is that of designating authorship credit to the right person for a piece of writing, such as a social media post or news article, found on the Internet.

We seek to implement a similar application of assigning authorship as a supervised classification problem, where the categories correspond to each of the possible authors. In our analysis, we will consider a dataset of snippets of writing from the Victorian era, a time widely considered to mark the start of the novel as the preeminent literary genre, at least in the English language. We will look at using logistic regression (using the glmnet package in R) [2] to correctly classify which of two authors wrote a given piece of text. To accomplish this, we will implement throughout the pipeline several of the benchmark methods of data analysis covered in class, namely matrix approximation/dimensionality reduction, regularized (logistic) regression, and cross-validation for tuning of regression parameters, to aid the construction of a robust and accurate classification model.

3 Data

The dataset used in this project is found on the UCI Machine Learning Repository [3] as the "Victorian Era Authorship Attribution Data Set." In addition, further description of the data collection process and the dataset itself are available from the publisher, but we present a brief breakdown here for completeness. The data consists of 1000-word snippets from Victorian era literary works, written by 50 distinct authors of the time period. A preview of the original data as collected can be seen in Figure 1 below. It should be noted that when splitting larger manuscripts to compile these samples, the dataset publisher chose to eliminate any words not appearing a significant number of times in the corpus; that is, only the more common words in the corpus as a whole were kept in the dataset. For instance, a unique or infrequent word like "frabjous" or "callooh" would likely not be included in the final dataset.

Originally, the dataset is used by the author for multi-class classification tasks [4], but we choose to narrow our focus to a binary classification problem so as to be able to effectively delve into our methods of interest stated earlier. So, for the purposes of this analysis, we take only a small subset of the data; we limit the number of authors at hand to two that appear interesting to cross-examine, Charles Darwin and Charles Dickens. These are two of the most prominent writers of the time (or of any time) and present somewhat differing contributions

Index	text
1057	broad moat towers use must oat end closely provisio…
987	admire ingenuity time nay pen fell hand wandering t…
1149	put one sleep said way encouragement finished thou …
980	charming weather visit mrs said playing suppose wal…
1135	allowed named art gave much fo became execution del…
957	much objection likes going yes disappointment never…
1271	appear like toys said must b thinking said john ah …
993	afternoon observed oddly mr behaved morning indeed …
954	foolish wish dear observed mrs happiness depends mo…
1013	first fit embarrassment fast yielding powers entert…
1158	born great things could speak look like could fly l…
977	quite sighed pinned placed sleeve frock sleeve wron…
978	persons whose station opinion stood less awe saw mi…
1420	masters ever painted angels thread great dear old l…
1277	girl changed teeth found last night put finger mout…

Figure 1: Preview of original data frame

to the era's literature; the task at hand could even be interpreted as a microcosm of a broader classification problem, such as nonfiction versus fiction, etc., but we stick to a simple "Charles versus Charles" examination. Their writings, at least to human intuition, seem contrasting enough to warrant being pitted against each other in a binary class structure.

In the end, there are 595 observations in total for us to work with, and although each writing sample consists of 1000 words, we choose to keep only the first 100 of these words for each observation in our analysis due to computational limitations. Here, we also perform a train-test split (choosing to keep 60% in the training set and 40% in the test set) for model assessment later. This gives us 357 training samples and 238 test samples. As we can see in Figure 2 and Figure 3, there is a slightly larger number of observations in the "Darwin" class (in both the training and test sets), but we proceed since this level of imbalance (60-65%) is typically not severe enough to warrant any concern in classification models.

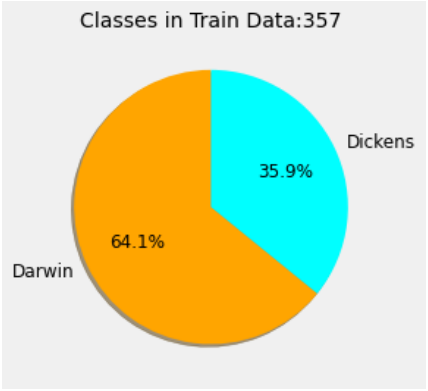

Figure 2: Breakdown of data in training set

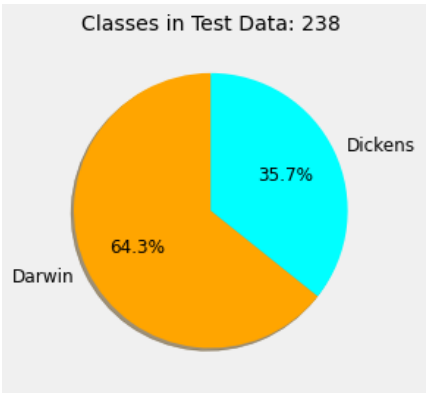

Figure 3: Breakdown of data in test set

We must next carry out a few steps to preprocess and transform the text data into a numerical data format, one that is suitable to be fed into models; our end goal in this phase is to take the raw text and somehow extract a numerical way of describing each data observation. In order to convert the data into a format ready for modeling, we proceed to apply several established techniques of natural language processing; this is done in Python (rather than R) as this language allows for much smoother handling of character strings and contains some convenient packages for preprocessing text in general. First, we implement tokenization, splitting the strings of continuous text (i.e. the writing samples) into lists of individual words appearing in those strings; the delimiter used in this case is the empty whitespace character ' '. Next, we want to remove "noise" from the text, meaning any components of the text that are not significant to the overall meaning; these include things such as punctuation marks, upper-case letters, numbers, and stopwords. Stopwords in English are words like "of," "the," "a," etc. The last step is stemming, or the removal of any affixes so that different words can be grouped together according to their root. After all, there should be no intrinsic difference in usage or meaning between, for example, "walked" and "walking." As a whole, in this stage, we have transformed the raw text of each writing sample into a list consisting of the (stemmed) individual words found in that sample.

Now that the text has been preprocessed, we must decide how to capture this data in a numerical way. There are many options available which are commonly used in natural language processing literature; we implement one of the most commonly used methods, Term Frequency-Inverse Document Frequency (*tf-idf*) vectorization. In *tf-idf*, a "weighted average" of sorts, measuring the number of times each word appears in the corpus, is assigned to each word according to the following formulas:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \epsilon d} f_{t',d}}$$

$$idf(t, D) = log\frac{N}{|\{d \epsilon D : t \epsilon d\}|}$$

$$tf - idf(t, d, D) = tf(t, d) \times idf(t, D)$$

where $f_{t,d}$ is the number of times that term t occurs in document d and the denominator for tf is the total number of terms in document d. Also, N is the total number of documents in the corpus and the denominator for idf is the number of documents where the term t appears.

Essentially, *tf-idf* assigns less importance to words which appear very frequently in the entire set of documents as a way of balancing/weighting the relative impact a word may have in an individual document. The final values of the vectorized words will always be in the range $[0, 1]$. At this point, after *tf-idf* vectorizing the text, we are now working with large (and more importantly, sparse!) numerical matrices. The new matrix still contains the same number of rows as before, but we now have several thousands of columns (3230 to be exact), each corresponding to each unique word found in the entire set of writings. An example of a matrix after tf-idf conversion can be seen in Figure 4 below, where we can immediately confirm that the vectorized data matrix is indeed extremely sparse as expected.

Index	aa	abandon	abash	abject	abl	abod	abroad	abruptli	absenc
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0.136741	0	0	0	0
10	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0

Figure 4: Preview of *tf-idf* vectorized data matrix

4 Analysis

4.1 Exploratory Data Analysis

Before jumping into model-building, we briefly present some exploratory plots that give a peek into the trends present in the data now that we have ways of representing of individual words as columns of a matrix. Figures 5 and 6 depict the most common words appearing in all of Darwin's writing and all of Dickens' writing, respectively. Some of the most frequent words do not at first glance appear all too revealing (e.g. many very commonplace words like "said" and "would" are common across both authors). However, upon a closer look, we can indeed see interesting patterns that play to our human intuition. For instance, words like "mother," "father," and "eye" are all quite frequent in Darwin's texts, which seems natural given that he was a pioneer in biology and genetics. So in the end, we can be both encouraged and puzzled regarding the possibility of differentiating between the writings of the two men. Either way, using statistical methods is a great way to go beyond simple intuition and to find the revealing patterns hidden in the data. [1]

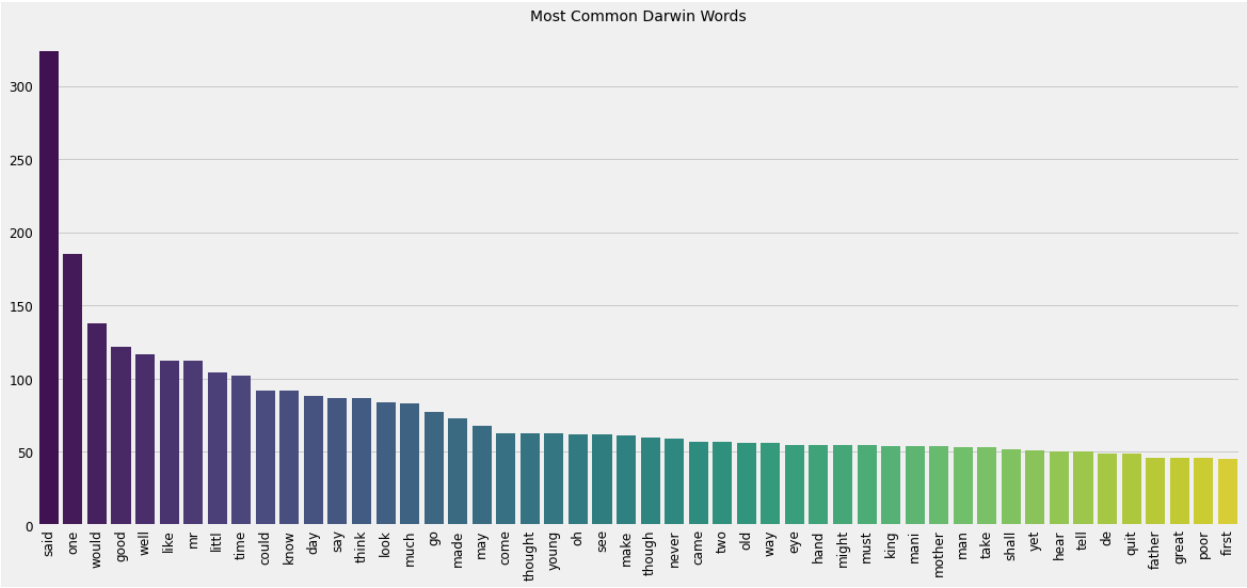

Figure 5: Most frequent words appearing in Darwin category

[1] As a side note, even at a relatively small sample size, the data seem to be following the well-known Zipf's law, a key result in quantitative linguistics.

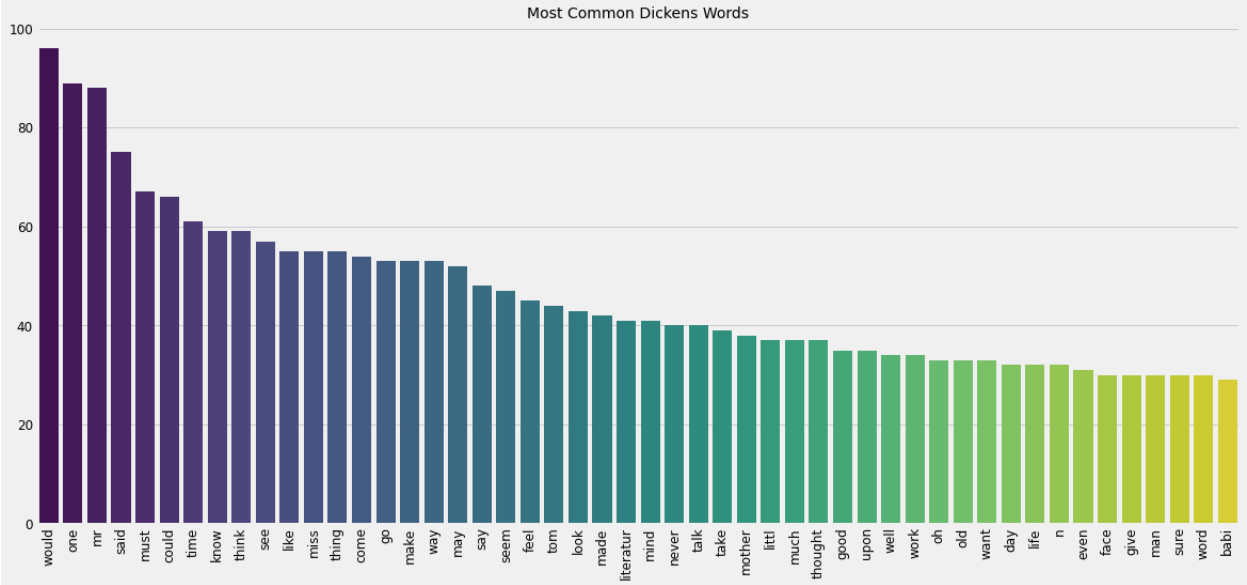

Figure 6: Most frequent words appearing in Dickens category

4.2 Logistic Regression

Since we are dealing with a binary classification task, we can start with a simple logistic regression model to serve as a baseline. Logistic regression, also known as the logit model, is an extension of linear regression to the case where the outcome is discrete and is one of the more popular and interpretable methods applied to binary classification tasks, used in many areas of modern statistical analysis, from biology to the social sciences.

We choose to use logistic regression for its simplicity and more importantly, to take the opportunity to take concepts covered in class for regular linear regression and extend them. We will measure the effectiveness of this and subsequent models using classification accuracy (i.e. the proportion of correctly classified observations). We implement the logistic regression using the glmnet package in R, and we elect to use regularized regression in order to enhance the prediction accuracy and reduce the risk of overfitting to the training data. The procedure for logistic regression using glmnet is the same as for linear regression, only with the added argument of `family = "binomial"` to indicate a proper link function for the generalized linear model. We run both a lasso and ridge regression, given by the least-squares minimization of the equations below: [2]

$$\sigma_{lasso}(\beta) = ||Y - X\beta||^2 + \lambda||\beta||$$

[2]When the regularization/penalty term λ is 0, the term goes away and we are left with the ordinary least squares problem.

$$\sigma_{ridge}(\beta) = ||Y - X\beta||^2 + \lambda||\beta||^2$$

The choice of the regularization constant λ can be fine-tuned to further bolster the model accuracy. We can first use cross-validation to find the λ that gives the minimum prediction error; then, we can use this value as the λ in the final model. We also try the one standard error (SE) above rule as an alternate scheme for choosing the optimal λ. Figure 7 gives a visual representation of this hyperparameter tuning process as an example.

We present the results of the lasso and ridge models both before and after cross-validated parameter tuning in Table 1. In addition, we present in Table 2 the run times for the cross-validation models from `cv.model` in glmnet. One thing to note is that sometimes, the choice mechanism for λ (minimum or 1 SE rule from CV) has no bearing on the final test accuracy; this is likely due in our case to a relatively small sample size, but there could exist other possible explanations to explore. Nevertheless, it is always interesting and good practice to consider multiple approaches to hyperparameter choice, so we leave in both sets of results for completeness. We note overall that the accuracy across the methods is quite high; this can serve as a benchmark for future models, when we are also worried about other aspects of model performance, such as overall simplicity/parsimoniousness and computational cost.

	Lasso	Ridge
Minimum	0.8823529	0.9327731
1 SE Above	0.8823529	0.9327731

Table 1: Regularized Logistic Regression Accuracy Results (from Full Data)

	Lasso	Ridge
Time (s)	3.16889	17.93844

Table 2: CV Model Run Times (from Full Data)

4.3 Singular Value Decomposition

Another crucial aspect of our problem to consider is that of high dimensionality, a common issue in statistical models, particularly in NLP applications. As previously mentioned, the *tf-idf* matrix we are using as the model matrix for the logistic regression model has essentially thousands of columns. While perhaps physical properties such as projection may not be so relevant in text classification, computational cost becomes a real concern. Although we are currently working with a somewhat modest-sized dataset with a few hundred obvservations, fitting a model can become resource-intensive and quickly get out of hand with more data.

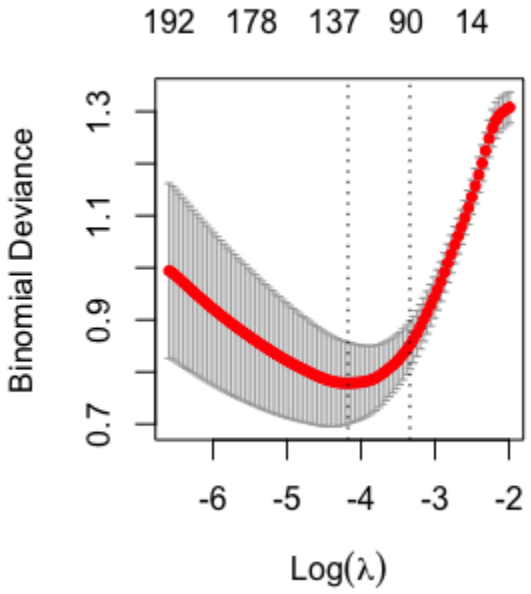

Figure 7: Hyperparameter tuning for regularized (logistic) regression

This is especially true in text classification since more observations would inevitably lead to many more unique words and hence more and more columns, in a deadly cycle.

To tackle this curse of dimensionality, we consider applying matrix approximation techniques to reduce the scope of the data and the computational cost. Since we are working with a numerical matrix, we have at our disposal a plethora of factor analysis methods as a means of attack. The most popular option is generally principal component analysis (PCA), in which we look for the first n principal components — an orthogonal set of vectors corresponding to linear combinations of the original explanatory variables — explaining a maximal amount of variation in the X variable. However, PCA runs into some issues when applied to sparse matrices such as the one we have here; this is a well-known computing issue in NLP as almost any vectorized matrix of word occurrences will be quite sparse. So, as an alternative, we instead look towards simply using the singular value decomposition (SVD) $X = U\Sigma V$, where U and V are orthogonal matrices and Σ is a diagonal matrix. In this factorization, we recall that the diagonal entries σ_i of Σ represent the singular values and the column vectors u_i and v_i of U and V correspond to the left and right singular vectors, respectively. It turns out that when we select the k largest singular values and their corresponding singular vectors from U and V, we can get the rank k approximation to X with the smallest error (via the Frobenius norm), mirroring the procedure in PCA.

More specifically, in the context of text classification, we can carry out a latent semantic analysis (LSA) [5], a technique built upon the SVD specifically engineered for sparse occurrence matrices. [3] In this context, from the SVD, we can now do things such as treat "term vectors" and "document vectors" as a "semantic space" using the new, lower-dimensional approximation of the higher-dimensional space. This and other related concepts are outside the scope of this study, but the main takeaway from the implementation of SVD/LSA is that we are able to open many doors in interpreting the dimensionality and other latent properties of the original word occurrence matrix, in addition to finding a more efficient low-rank approximation of the original, computationally demanding matrix.

We implement LSA using the function TruncatedSVD in the scikit-learn (sklearn) [6] package in Python, which unlike a regular SVD procedure, produces a factorization where the number of columns can be specified for a number of truncation. This transformer function performs a linear dimensionality reduction but contrary to PCA, this estimator does not center the data before computing the singular value decomposition; this means it can work with sparse matrices efficiently. For our analysis, we choose to keep the first 100 components using TruncatedSVD; we can visualize the amount of variance explained by the first n components using the scree plot in Figure 8. Of course, in practice, one may use cross-validation or other techniques to search for an optimal number of components to keep, however we do not place the focus of our analysis on this and rather choose to simply use 100 (as recommended by the TruncatedSVD documentation for most applications). While it appears that the first few components do not cover as much of the variance as one might similarly expect from say, PCA, it should be noted that preserving the variance is not the exact objective of truncated SVD without centering. In fact, as we will see later, the relatively small amount of variance explained by 100 components does not apparently have a profound negative impact on modeling. In the end, the 100 components we keep explain roughly half of the variance in the X data. Considering the original matrix actually consisted of 3,230 columns, this is quite a significant cutback. We can see as in Figure 9 that the new, decomposed matrix after the truncated SVD is no longer sparse and looks much more akin to what we commonly see in regression modeling contexts.

By using this low-rank approximation to the full data, we are able to reduce the dimensionality of the data (and hence its size and storage burden) by several orders of magnitude. Next, we can use this new, reduced data to build alternate models to compare to the earlier,

[3] LSA "mitigates the problem of identifying synonymy, as the rank lowering is expected to merge the dimensions associated with terms that have similar meanings. It also partially mitigates the problem with polysemy, since components of polysemous words that point in the 'right' direction are added to the components of words that share a similar meaning."

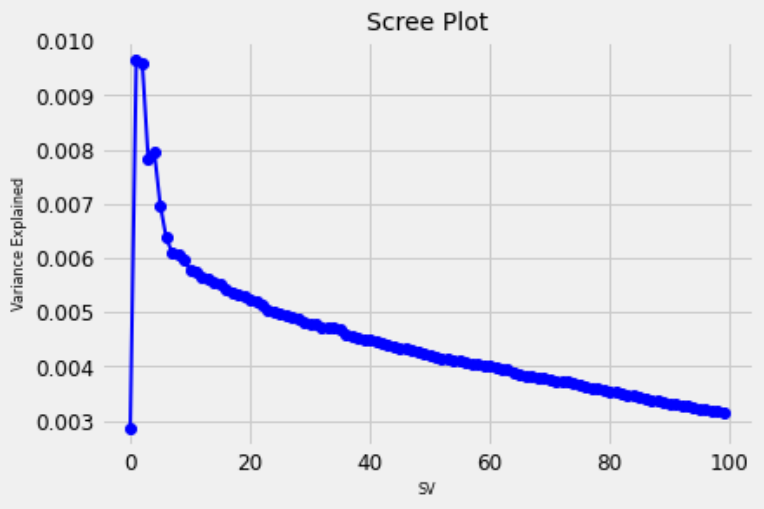

Figure 8: Scree plot for Truncated SVD

"full data" models. As before, we implement a lasso and ridge regression and use cross-validation to tune the regularization term in both cases. In this new round of modeling, the program runtime is significantly reduced due to the new, smaller size of the data matrix. The accuracy results are presented in Table 3 and offer some surprising insights when compared to the corresponding values in Table 1. In addition, the run time results are presented in Table 4 and offer a stark contrast to the high values we saw earlier when using the complete data for modeling.

	0	1	2	3	4	5	6	7	8	9
0	0.0881495	0.035314	−0.02422…	−0.08107…	0.0573879	−0.00918…	−0.02678…	0.0961356	−0.00800…	−0.02705…
1	0.225027	0.0220533	−0.03689…	−0.09222…	0.110784	−0.05180…	−0.039751	0.093966	−0.06744…	−0.138302
2	0.205842	−0.109187	−0.08621…	0.0244767	0.0166942	−0.06297…	−0.110859	−0.124024	−0.106016	0.0760925
3	0.232567	0.0277617	0.128183	0.0187457	0.000309…	−0.04850…	0.0297161	0.0635554	0.0447978	0.030297
4	0.133299	−0.02786…	−0.09714…	−0.188326	0.0399793	−0.01774…	0.083886	−0.08749…	0.132906	0.163309
5	0.230374	−0.02316…	0.0933624	−0.05901…	0.0136951	−0.04473…	−0.00186…	0.084586	0.0522912	−0.06425…
6	0.227299	−0.09910…	−0.00499…	0.129991	−0.06122…	−0.117861	−0.05124…	−0.00095…	0.0382337	−0.05809…
7	0.217667	−0.00565…	0.197521	−0.07707…	0.12899	0.132506	0.0720935	−0.04917…	0.0603683	−0.039029
8	0.172057	0.0203245	0.107095	0.0190546	0.0448839	0.0363408	0.0348543	0.0675568	0.158209	−0.06082…
9	0.16213	−0.00490…	0.0181493	−0.02116…	−0.01935…	−0.06286…	−0.126557	−0.01643…	−0.00986…	−0.00357…
10	0.258071	−0.09977…	−0.02593…	0.0557047	−0.07438…	−0.07627…	0.0289585	−0.01268…	0.0920217	0.103601
11	0.14847	0.0805528	0.119001	0.004504…	0.0741818	0.0116645	−0.00246…	0.0540533	0.0257919	0.058162
12	0.126379	0.0502364	0.0973066	−0.03227…	0.0929396	0.007905…	−0.04003…	0.0453534	0.003336…	0.0349891
13	0.227392	−0.04520…	0.002748…	−0.109604	−0.132191	0.0595244	−0.02984…	−0.01781…	0.0893806	−0.00698…
14	0.203958	−0.07749…	0.045033	0.0513888	−0.06506…	0.0323789	0.0151484	−0.08801…	−0.110333	−0.09818…

Figure 9: New, reduced data matrix after LSA

	Lasso	Ridge
Minimum	0.9117647	0.907563
1 SE Above	0.8655462	0.9243697

Table 3: Regularized Logistic Regression Accuracy Results (from Reduced Data After SVD)

	Lasso	Ridge
Time (s)	0.6832712	0.6874232

Table 4: CV Model Run Times (from Reduced Data After SVD)

5 Discussion

As we have seen, we are able to preserve and even improve upon the results from running logistic regression using the whole data by first reducing the dimensionality of the *tf-idf* occurrence matrix via a truncated singular value decomposition (also known latent semantic analysis in the context of natural language processing). This low-rank matrix approximation brings many benefits when it comes to handling the large word occurrence matrix common to all text classification problems, chiefly the large reduction in computational cost and data storage requirements. It is an excellent solution to circumventing the curse of dimensionality, which causes algorithms to struggle to converge to an effectively trained model.

The classification accuracy after applying the truncated SVD does decrease, as we can see in the tables below, however not by a very substantial amount. This is especially unconcerning when we also compare the computing time and resources necessary to train and build the models before and after dimensionality reduction. In addition, sometimes, the data dimensionality reduction may sometimes even help to improve the accuracy of a model, as happened in some pre-experimental models not presented as final results in this paper. This highlights the fact that SVD allows us to untangle and extract latent information present in high-dimensional data by removing irrelevant features that may lead the model to be based on insignificant factors and as a result, less accurate.

Though, even in a case like this one where accuracy may slightly diminish after a matrix approximation, it is often still worthwhile to accept the trade-off between predictive power and computational resources/demands. Although we used SVD/LSA, this is akin to the typical procedure in PCA, where we may select the first n principal components which may still capture up to 95% (or a similar cutoff percentage) of the variation. The dimensionality in a text-based dataset of this form may very easily get out of hand; even with only a few hundred 100-word observations, the model building becomes noticeably faster after applying the SVD. One can imagine, then, the sort of importance that dimensionality may play if dealing with say, a dataset of hundreds of thousands of observations, or a dataset where

each observation is itself is an entire manuscript and not just 100-word samples. Properly dealing with the high dimensionality and accompanying data storage concerns is certainly a very real problem due to the nature of text data.

Of course, the LSA methodology is not the only choice we have for addressing the curse of dimensionality. While as mentioned, regular PCA is not always appropriate for sparse matrices such as a *tf-idf* (or other vectorized) matrix common in text classification, there is currently much ongoing research into the topic. For example, Reza Drikvandi1 and Olamide Lawal [7] propose a "sparse PCA" alternative to ordinary PCA in their paper, *Sparse Principal Component Analysis for Natural Language Processing*, promising "two major advantages, faster calculations and easier interpretation of the principal components."

All in all, the problem of high dimensionality is one that deserves major emphasis in all of statistical modeling, but especially in the world of natural language processing. As we have covered, the nature of tasks such as text classification lead to sparse matrices that are often enormous in size, which warrant finding new and efficient solutions to the resultant slow or even ineffective predictive modeling procedure. While we have only focused here on latent semantic analysis, this is an exciting problem at the intersection of statistics and computing today, and more effective methods such as LSA will surely continue to be proposed as a way to tackle this difficult yet important question.

6 References

[1] Mosteller, F., & Wallace, D. L. (1963). Inference in an authorship problem: A comparative study of discrimination methods applied to the authorship of the disputed Federalist Papers. *Journal of the American Statistical Association*, 58(302), 275-309.

[2] An Introduction to glmnet. *CRAN R Repositary.*

[3] UCI Machine Learning Repository: Victorian Era Authorship Attribution Data Set. https://archive.ics.uci.edu/ml/datasets/Victorian+Era+Authorship+Attribution

[4] Gungor, A. (2018). Benchmarking authorship attribution techniques using over a thousand books by fifty Victorian era novelists.

[5] Landauer, T. K., Foltz, P. W., & Laham, D. (1998). An introduction to latent semantic analysis. *Discourse processes*, 25(2-3), 259-284.

[6] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12, 2825-2830.

[7] Drikvandi, R., & Lawal, O. (2020). Sparse principal component analysis for natural language processing. *Annals of data science*, 1-17.