STAT 154: Modern Statistical Prediction and Machine Learning Final Project Report (Team 14)

Eddy Zhao[1,2], Dotun Lytton-Fatade[1,2,3], Kuang Li[1,2] & Hoon Kim[1]

December 2, 2016

[1]Department of Statistics, University of California, Berkeley
[2]Department of Economics, University of California, Berkeley
[3]Department of Mathematics, University of California, Berkeley

Abstract

In this report, we investigate and tune an e-mail classification model using three statistical learning algorithms: Random Forests (RF), Support Vector Machine (SVM) and Kmeans Clustering (KMC). Sparse learning supervised feature selection technique is implemented on the training set, which allows to embed feature selection into the classification.

1 Introduction

On March 2, 2015, The New York Times (Schmidt 2015) broke the story of how Secretary Clinton had exclusively used a private email server rather than a government-issued one throughout her time as Secretary of State, and that her aides took no action to preserve emails sent or received from her personal accounts as required by law. This has led to an ongoing political controversy the likes of which havent been scene in US politics since the Penny Debate of 1990 (HR 3761-1989). While the subject of this analysis, the Clinton Emails, can be seen as controversial, it highlights the challenges of dealing with real-world, unstructured and open publicly available data. An e-mail is composed of date, e-mail address, subject, body of the e-mail, and so on. It is possible for the body to include pictures, sounds, and programs, but the body is mainly composed of textual data. Thus, it is possible to use text mining techniques to analyze e-mails. Accordingly, text classification in this context has become one of the more important issues in text mining and information retrieval. The main objective of this project is to develop a machine learning classifier to correctly classify texts to the e-mail sender according to their attributes as accurately as possible. There are two main types: Binary Classification and Multi-class Classification. While automated text classification involving modern machine learning technique is still in its infancy, it has been successfully applied to many applications including text mining. More pertinently, it can be

used as an efficient method for preliminary content inspection and rating. In this report, we present three models developed using three machine learning algorithm: Random Forest, Support Vector Machine and Kmeans Clustering. The performance of these methods are compared to each other. In addition, to foster reproducible data science, the methodologies and performance of the various unsupervised and supervised selection methods implemented on the word feature selection and the optimal tuning parameters for Random Forest, Support Vector Machine and Kmeans Clustering are presented.

2 Data and Software

2.1 Data Description

The raw training dataset is composed of Hilary Clintons 3505 released personal emails from top 5 contacts among the full email dataset. Each of the top 5 contacts is labeled with a number and labels are simply: 1, 2, 3, 4, 5. Most emails share considerable amount of common words and noisy contents that may or may not help our classification model. We have enclosed a short summary statistic in Table 1.

Class Label	Total E-mail Count	Total Characters Count
1	676	1,311,956
2	1,037	2,381,892
3	1,234	1,344,507
4	271	284,759
5	287	1,985,534

Table 1: E-mail Summary Statistics

2.2 Stop Words

Stop words refer to the most common words in English such as I, me, am etc. To create a stop words vector, we utilized the stop words provided from the built in stop words vector in tm package as well as the stop words we obtained from online in which we saved as a .txt file and then loaded into RStudio as a vector. We appended the two vectors and filtered out repeated stop words to get our final stop words vector. The list of stop words is shown in Appendices, A.1.

2.3 R Packages

Our team chose R as the main programming tool to build classification models. In the data importing and cleaning process, we used R package tm. To run the random forest and support vector machine we utilized both randomForest and e1071 packages. While looking

for the best tuning parameter we implemented the Caret package. Last but not least we also used common packages such as ggplot2, stringr and stringi for plotting graph and further assist text mining.

3 Filtering & Feature Selection

3.1 Filtering

The first step we took after obtaining the raw training data was to filter out all the unnecessary noisy information contained in the data; Information such as punctuations, numbers, security clearance, and stop words that will not provide us much value when building classification models. To do so, we transformed all the emails to lower case letters then we used for loop and regular expression on each individual email to remove all the punctuations, numbers, any word with only single or double letters, and extra while spaces. To remove stop words, we utilized the removeWord function which takes in the stopwords vector we created earlier as argument and remove any word contained in the vector from the emails. Last but not least, to filter out the security clearance, common sentences and other common words in general, we used the gsub function to delete them, the specified sentences and words we removed are listed in Appendix B.

3.2 Word Feature Selections

We stored the train dataset into a Corpus with each individual email as a text document. Afterward, we used the common text-mining approach to create a term-document matrix from the Corpus. While implementing the DocumentTermMatrix function we set stemming TRUE to reduce the parsed words to their root form to avoid issues with typos made by the senders. By converting the output to matrix and then data frame enables us to obtain the word features and number of appearances for each email.

Nevertheless, the high dimensional feature space and sparsity of data became a potential issue. After several trial and error, we reduced the sparsity of data to 99.2% which yields the lowest error rate as we can see at the Appendix C. Furthermore, we applied stemming to each document term matrix to reduce each word to its root form to avoid spelling errors done by the senders.

We conducted the exact same approach to our test dataset except we did not apply removeSparse to the test document term matrix since we intend to keep as many test features as possible. Once the word features are extracted from the test dataset, we used the overlapping word features between the training and test data as our final features. In the end our final feature matrix contains 2036 features.

3.3 Power Feature Selections

To further enhance our datas accuracy, we implemented power features that will be helpful when determining the senders. We started by creating power features that would distinguish one senders style of writing from the others. We looked at the number of exclamation marks and question marks used for each email. Both of which can be counted using the *strcount* function. We also considered that different senders write in different lengths, use different choice of vocabularies, and different number of sentences. With the help of the *stringr* package, we obtained features for each email on number of sentences used, number of characters, and distinct words.

We also reasoned that different senders send out more emails on different days of the week and in different months of the year. Therefore, we also created new power features that shows which day and month each email is sent by counting the first encounter with words that represent week days and months as the sent date for that email.

Table 2 shows a brief metrics that tells how the number of features varied as we take different steps to make our features more accurate.

Steps	Brief Description	Total Features
1	Raw: the original training data file	73,375
2	Filtering: rids common sentences and words (other than stop words) in emails	36,244
3	Remove Stop Words: deletes all the stop words	36,112
4	Stemming: reduces words to its root form	26,139
5	Remove Sparse Terms: removes words sparser than certain percentage	2,024
6	Remove Non Overlapping Words: removes words not present in the test data set	2,005
7	Add Power Features: appends power to existing features to create a final feature matrix	2,036

Table 2: Final number of features metric

4 Models

4.1 Random Forest

For our Random Forest Model, we focused on tuning for the optimal number of trees while simultaneously tuning for the optimal *mtry* value. Although we used the caret package to cross validate for our feature selection, computational constraints led us to manually tune for *mtry* and forst size. We ultimately decided that the optimal value for *mtry* would be $p/3$ and that the optimal value for forest size would be 200 trees. In order to control for computational limits, we limited the range of trees to be either 50, 75, 100, 150, 200, or 300,

and the range of *mtry* to be either $\sqrt{p}$, $p/2$, and $p/3$.

The average OOB error rates for each value of *mtry* along with the individual class error rates are shown in the table below. We see from this table that the overall average error rate was lowest for when $mtry = p/3$. Subsequently, we find in the second table, that the lowest error rate among the models using $mtry = p/3$ corresponds to the model that uses 200 trees.

The error rate for each selection of forest size by *mtry* is shown below as well, in addition to the average error rate for each selection of forest size across all three values of *mtry*. The plot of this table is shown as well. We see that the forest size with the lowest average error rate across all three values of *mtry* was 200. Subsequently, the lowest error rate amongst the models utilizing 200 trees was the model with $mtry = p/3$.

We thus selected $p/3$ and 200 as the optimal values for *mtry* and the number of trees respectively, since we arrive at this conclusion whether we start with looking at *mtry* or number of trees. Had we encountered a situation where this was not the case, we would have picked the values corresponding to the model with the lowest overall error rate as our optimal values for *mtry* and number of trees.

After selecting the Random Forest Model with the optimal parameters, we manually used 5-fold cross validation to select a best model using the optimal parameters. We split the entire training dataset into five subsets and created five new datasets, each excluding a different subset. We then fit the Random Forest Model with the optimal parameters on each of the new datasets, and obtained predictions for the respective held out sets. After comparing the predictions of each model with the true classes from the held out data, we selected the Random Forest Model with the minimum error rate as our final Random Forest Model.

As with our Random Forest Model, our goal was to find the optimal kernel, cost, and gamma parameters. Due to computing constraints, we again manually validated for the optimal kernel by taking a random sample of 1500 observations from the training data and fitting an SVM with a linear, polynomial, and radial kernel. We then tested each model against the held out data for model accuracy. The confusion matrices below indicate that the linear kernel performed the best, and so we continued our validation on the linear kernel SVM only.

Number of Trees	50	75	100	150	200	300
A: m=$\sqrt{p}$	0.284	0.274	0.273	0.262	0.260	0.258
B: m=$p/2$	0.258	0.251	0.258	0.250	0.254	0.253
Mean Error Rate	0.267	0.260	0.261	0.254	0.254	0.254

Table 3: Cross Validated Errors

Average Error Rate	OOB	Class 1	Class 2	Class 3	Class 4	Class 5
A: m=$\sqrt{p}$	0.269	0.565	0.218	0.152	0.357	0.169
B: m=$p/2$	0.254	0.510	0.223	0.149	0.322	0.150
C: m=$p/3$	0.253	0.510	0.217	0.153	0.319	0.143

Table 4: Average Cross Validated Errors

4.2 Support Vector Machine

Using the caret package once more, we tuned our SVM using a linear kernel, and a range for cost and gamma. We limited cost to the values of .1, 1, and 10 and gamma to the values of 0.5, 1, and 2, due to computational constraints. We decided these values based off similar examples in ISLR. As shown in the table 5, the optimal values of cost and gamma that we obtained from tuning were 0.1 and 0.5 respectively, with an error rate of 0.270.

Finally, after obtaining the optimal parameters for our SVM model using a linear kernel, we fit such an SVM on our entire training set and ran a summary to assess our model. The final model included 1754 support vectors. Other relevant results are shown in the overall summary.

	Cost	Gamma	Error	Dispersion
1	0.1	0.5	0.270	0.011
2	1	0.5	0.314	0.020
3	10	0.5	0.348	0.021
4	0.1	1	0.270	0.011
5	1	1	0.314	0.020
6	10	1	0.348	0.021
7	0.1	2	0.270	0.011
8	1	2	0.314	0.020
9	10	2	0.348	0.021

Table 5: SVM Tune Performances

	True 1	True 2	True 3	True 4	True 5
1	377	3	0	0	1
2	4	584	4	0	0
3	11	12	690	1	1
4	0	0	0	156	0
5	0	0	0	0	161

Table 6: Linear Kernel Error Rate

	True 1	True 2	True 3	True 4	True 5
1	272	2	1	0	3
2	3	489	3	0	2
3	115	108	690	56	12
4	0	0	0	101	0
5	2	0	0	0	146

Table 7: Polynomila Kernel Error Rate

	True 1	True 2	True 3	True 4	True 5
1	47	4	6	5	4
2	79	273	88	26	36
3	266	322	599	122	43
4	0	0	0	3	0
5	0	0	1	1	80

Table 8: Radial Kernel Error Rate

4.3 Kmeans Clustering

For our unsupervised method of K-Means clustering, we naturally approached differently from our supervised methods. In order to achieve best classification accuracy, we imported the top 100 important features from our best Random Forest model, and filtered the data set with those top 100 features. In addition to filtering the data set with the top 100 features, we normalized the data set (such that values would take on z-scores) to minimize the bias. The tuning parameter we sought to optimize was *nstart*, or the number of iterations of the K-Means clustering algorithm that would be run before the function decided on the optimal model, using the default distance metric (distances from each centroid) and the error rate, which we obtained by comparing the cluster value of each point to the corresponding response variable.

For our unsupervised method of K-Means clustering, we naturally approached differently from our supervised methods. In order to achieve best classification accuracy, we imported the top 100 important features from our best Random Forest model, and filtered the data set with those top 100 features. In addition to filtering the data set with the top 100 features, we normalized the data set (such that values would take on z-scores) to minimize the bias. The tuning parameter we sought to optimize was nstart, or the number of iterations of the K-Means clustering algorithm that would be run before the function decided on the optimal model, using the default distance metric (distances from each centroid) and the error rate, which we obtained by comparing the cluster value of each point to the corresponding response variable. In order to validate the best nstart value, we conducted 5 fold cross-validation with range of nstart values from 20 to 100. After cross-validation, we were able to observe that the optimal nstart according to the error rate was 80 and the optimal nstart according to the distance metric was 20. With the optimal nstart values found from the cross-validation process, we conducted K-Means clustering classification on the full training data set. K-Means Clustering analysis on the full training data using the optimal nstart value according to the error value reported 71.67% accuracy, whereas the same analysis using the optimal nstart value according to the distance metric reported 79.12% accuracy. From this result, we decided that the distance metric has better performance in prediction.

Since distance metric showed better performance in prediction, we decided to conduct cross-validation on the test data set for optimal nstart value using distance metric as our error measure. From the cross-validation, we found 100 as the optimal nstart value for the test data set. Interestingly, the optimal nstart value for test data set found from cross-validation showed discrepancy from that of the training data set. After attaining the optimal nstart value for the test data set, we employed it to predict the classes of the test data set.

5 Summary

Presented below are the classification model summaries.

Step	Total # of features used	Total Accuracy	Accuracy per sender class
SVM	2035	49.96%	1:46.44% 2:60.56% 3:39.38% 4:35.79% 5:78.37%

Step	Total # of features used	Total Accuracy	Accuracy per sender class
Random Forest	678	.7404	1: .4705 2: .7865 3: .8459 4: .6528 5: .8431

6 Discussion

Throughout the process of feature and model selection, there were several potential sources for improvement. Possible setbacks that we could have controlled for involved creating our final feature matrix. Due to sheer volume of potential features, there is a possibility that the feature space we ultimately selected either overfit or underfit our data. The methods we used to parse our data as well as the choices of which words to include and exclude could have introduced an implicit bias in our models as well, since one cannot tell from the data parsing step, which features will actually have the most impact when utilized in models. Furthermore, though we capitalized on many power features, the list for potential power feature candidates is endless, and it is likely that there were patterns and trends that we simply did not notice or account for.

Additionally, there were several sources for improvement that were not in our control. For example, we could have made use of the ROC curves to assess model strength, in addition to the metrics we chose. There were also methods that were not required, but could have contributed to better results, such as Gradient Boosting and other forms of unsupervised clustering. Finally, there were methods not discussed in the scope of this task, such as Neural Networking, which could have potentially acted as a better classifier or provided more insight into the classification task at hand.

7 References

[1] Edward Loper and Steven Bird. "NLTK: The Natural Language Toolkit". In: *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*. ETMTNLP '02. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 63-70. DOI: 10. 3115 / 1118108. 1118117. URL: http: //d.x.doi.org/10.3115/1118108.1118117.

[2] G. James, An introduction to statistical learning, 1st ed.

8 Appendices

8.1 A. Word Feature Stop Words from Online

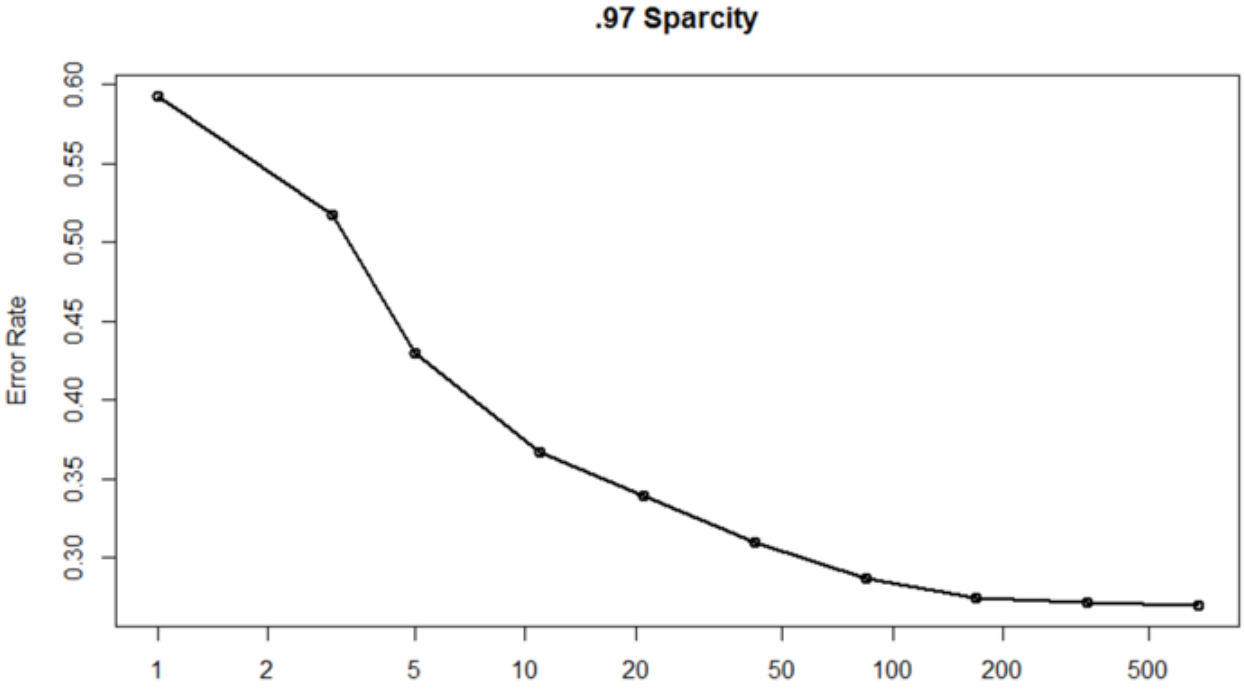
[1,] "a" "do" "its" "said" "were" "those" "we'll" "where's" [2,] "able" "does" "just" "say" "what" "being" "they'll" "why's" [3,] "about" "either" "least" "says" "when" "having" "isn't" "how's" [4,] "across" "else" "let" "she" "where" "doing" "aren't" "until" [5,] "after" "ever" "like" "should" "which" "ought" "wasn't" "against" [6,] "all" "every" "likely" "since" "while" "i'm" "weren't" "between" [7,] "almost" "for" "may" "so" "who" "you're" "hasn't" "through" [8,] "also" "from" "me" "some" "whom" "he's" "haven't" "during" [9,] "am" "get" "might" "than" "why" "she's" "hadn't" "before" [10,] "among" "got" "most" "that" "will" "it's" "doesn't" "above" [11,] "an" "had" "must" "the" "with" "we're" "don't" "below" [12,] "and" "has" "my" "their" "would" "they're" "didn't" "up" [13,] "any" "have" "neither" "them" "yet" "i've" "won't" "down" [14,] "are" "he" "no" "then" "you" "you've" "wouldn't" "out" [15,] "as" "her" "nor" "there" "your" "we've" "shan't" "over" [16,] "at" "hers" "not" "these" "myself" "they've" "shouldn't" "under" [17,] "be" "him" "of" "they" "ours" "i'd" "can't" "again" [18,] "because" "his" "off" "this" "ourselves" "you'd" "couldn't" "further" [19,] "been" "how" "often" "tis" "yours" "he'd" "mustn't" "once" [20,] "but" "however" "on" "to" "yourself" "she'd" "let's" "here" [21,] "by" "i" "only" "too" "yourselves" "we'd" "that's" "both" [22,] "can" "if" "or" "twas" "himself" "they'd" "who's" "each" [23,] "cannot" "in" "other" "us" "herself" "i'll" "what's" "few" [24,] "could" "into" "our" "wants" "itself" "you'll" "here's" "more" [25,] "dear" "is" "own" "was" "theirs" "he'll" "there's" "such" [26,] "did" "it" "rather" "we" "themselves" "she'll" "when's" "same"

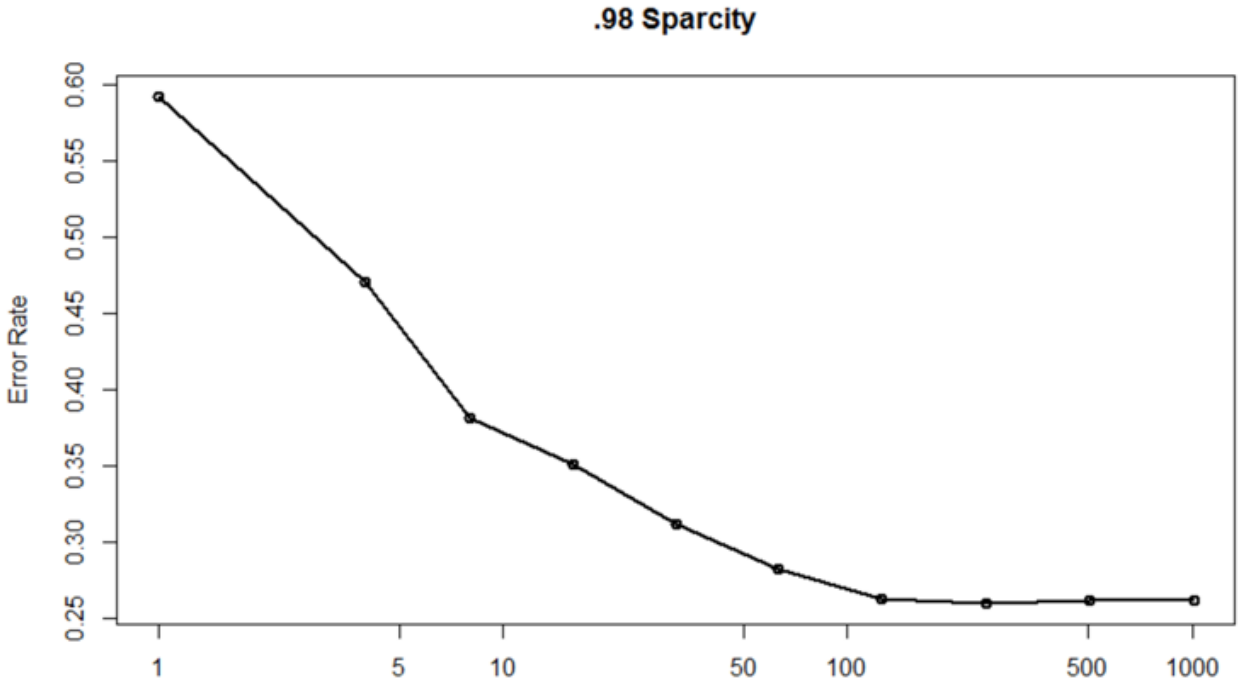
8.2 B. Specified Sentences and Words Removed

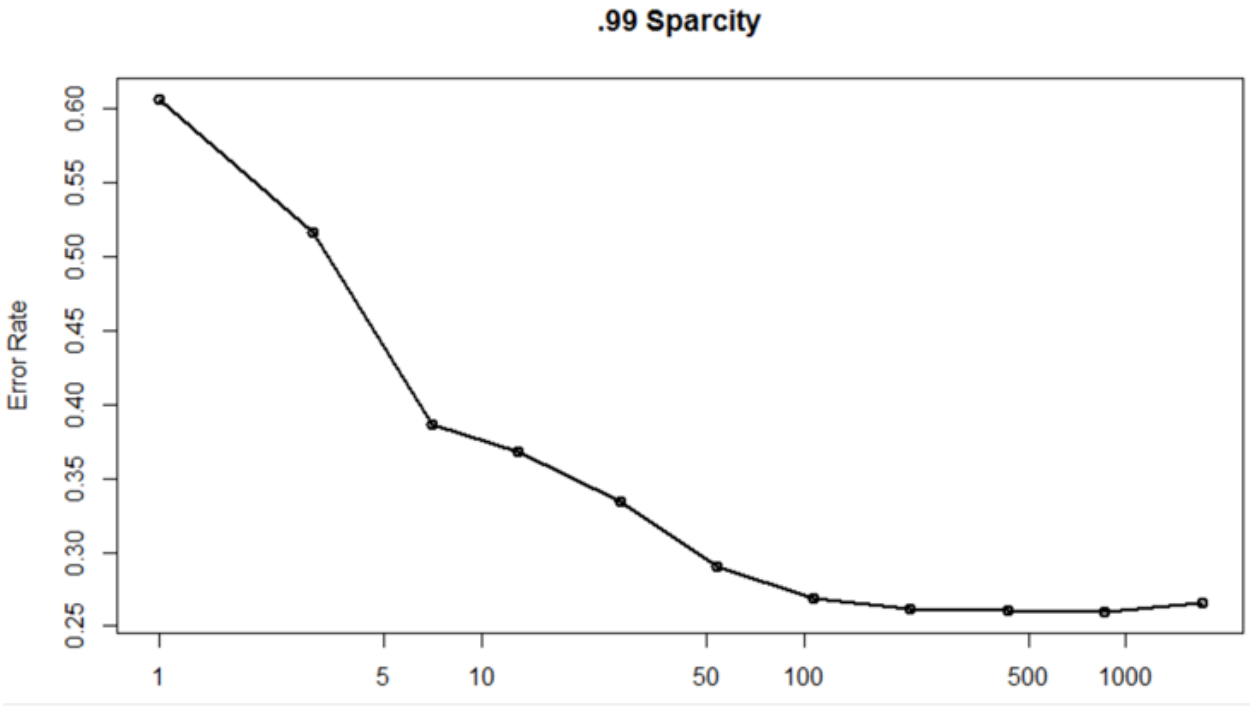
"unclassified u.s. department of state case no.", "doc no.", "date:", "state dept. - produced to house select benghazi comm. subject to agreement on sensitive information & redactions.", "no foia waiver.", "subject:", "sent:", "monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday", "f-", "release in full", "release in part", "state-", "january", "february", "march", "april", "may", "june", "july", "august", "september", "october", "november", "december", "subject", "fvv", "sent", "scb"

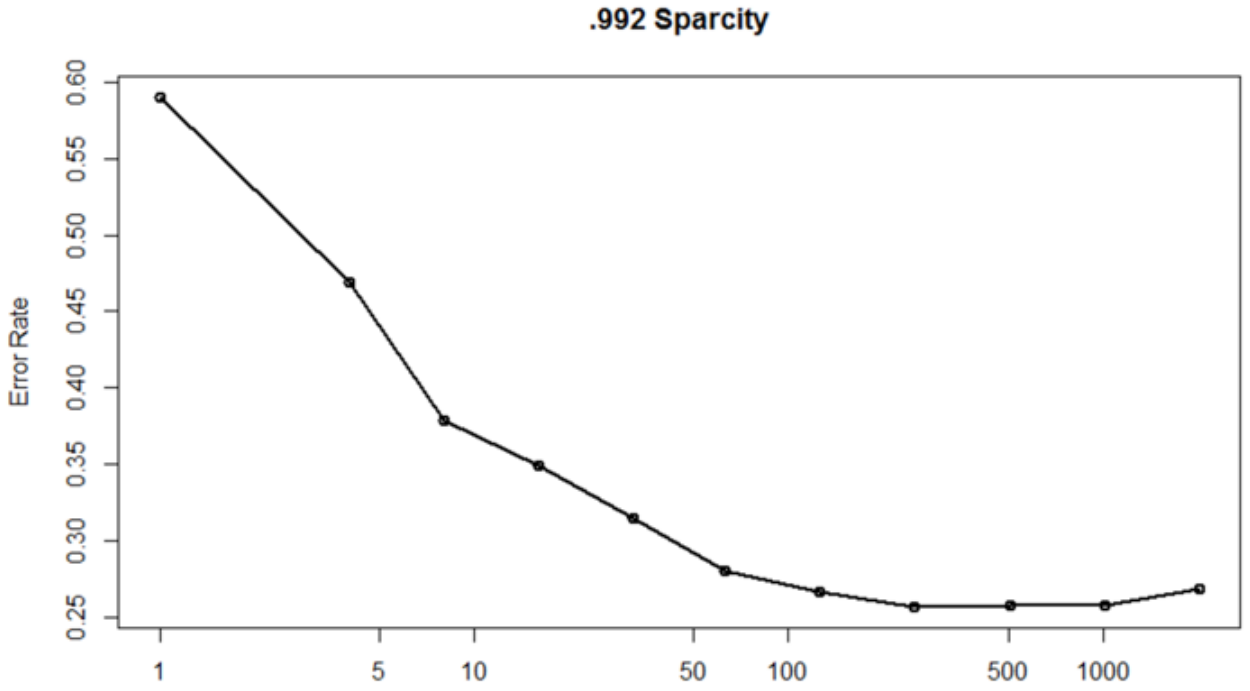
8.3 C. Sparsity Selection

*Note: The x-axis represents the number of predictor selected at each split. We can clearly see that 0.992 performs at a better rate.









8.4 D. R Program Code

Please check the github page