A Comparison of Text Classification Classifiers and Feature Transformations with the Federalist Papers

Ian Parr
University of Washington
ianparr@cs.washington.edu

Fentahun Reta
University of Washington
retaf@cs.washington.edu

Abstract

The Federalist Papers are a set of 85 essays that were written between October 1787 and August 1788. The principal authors are known to be as Jay, Hamilton and Madison. However, there are twelve documents whose authors have not been identified. In this paper, we attempt to classify these remaining documents into four categories: written by Madison, Jay, Hamilton, or a collaboration between Madison and Hamilton. We compare different classifiers and data transformations and discuss the results of using these different models.

1 Introduction

Document classification has a lot of applications of in the real world, such as grouping documents and extracting information, email spam filtering, identifying duplicate documents, author detection etc. However, due to the rapid growth of data and online information, it is no longer feasible for humans to manually observe data or to make categorical classifications. There are many different methods and models used for document classification. In this paper, we use a multitude of different classifiers which are: a Naive Bayes classifier, Complement Naive Bayes (CNB), k-Nearest Neighbors (k-NN), and a simple ensemble classifier. We also use a variety of data transformations, which are: simple word counts, tf-idf, and singular value decomposition (SVD). We apply these methods and transformations on the Federalist Papers. These consist of eighty-five essays that were written by Alexander Hamilton, James Madison, and John Jay, where the authorship of twelve are disputed. We attempt to classify these twelve essays using these different models, and compare the results.

2 Motivation

We compare these different methods for the reasons that text data is not uniform (since different documents will not be the same length) and that the documents will not be evenly distributed in terms of group classification. For the 73 papers whose authors are known, 50 are by Hamilton. This means that we have a data set which is unbalanced, since the majority of the papers are by Hamilton, and thus the documents are not evenly distributed. Furthermore, the papers are of varying length: for example, Federalist Paper 50 is about half as long as paper 73. This means that our models have to deal with skewed data and document length biases. Therefore, we will show in this paper that for a "good" classifier of text-data, it is necessary to implement transformations which deal with these biases.

3 Raw Data Processing

The papers were stored in a text file containing them concatenated together and also labeled. We used simple text processing to split up the file and labels and divided them up into the training and test set. The training set refers to the 73 papers whose authors were known and the test set were the 12 other 'disputed' papers.

With each individual paper now divided into these two categories, we initially want to form a document-term matrix for the both the training and test set for word counts. This is matrix whose columns consist of the unique individual words which occur in the training set. The reason for this is because the classifiers we use learn via supervised learning, which requires only the training set is used. We then transformed the text to all lower-case and removed punctuation. We got the number of unique words in the training set to be 8248. The vectors in the document-term matrix are just the count of that word in the document. We applied this to both the training and test sets to establish two matrices. This kind of matrix is also referred to as the "bag-of-words" model, since it does not take into account the grammar and structure of the documents.

4 Data Transformations

From this document-term matrix, we may directly train our classifiers, or apply feature transformations and reductions which may improve the performance of our classifiers. From this initial matrix, we have the problem that the data weights are not normalized, and also that we may have many features which are unimportant whose impact should be lowered or removed.

4.1 Feature Reduction

We explored feature reduction techniques which reduce the number of columns in our document-term matrix. This is done for two reasons. The first is that reducing the number of features will improve the runtime of our learning algorithms, as we will have less features to train our data on. The second reason is that such techniques may improve the accuracy of our classifiers. This is due to what's called the "big p small n problem" [6].

The first technique we used was to remove all "stop words" from our documents. This was done before we created our document-term matrix. "Stop words" refers to common words in the language we're analyzing (for example, "the", "and", "a") and thus are irrelevant when it comes to analyzing the data.

The second technique was variance filtering. This is done to remove all "low variance" words from our matrix. This is an extension of removing stop words, in that common words which occur in every matrix are removed. But this also applies to uncommon words which may, for example, only appear once in one document. This technique was applied after we had done the tf-idf transformation.

The third was Singular Value Decomposition (SVD). This is a technique from linear algebra which operates on a matrix that has been transformed by tf-idf. It works by grouping similar features on sparse matrices. When applied on a document-term matrix, this process is called Latent Semantic Analysis (LSA). By grouping similar features, we transform our feature space of words to a "concept space".

4.2 Feature Transformation

As discussed above, to apply variance filtering and SVD, we used the tf-idf (term frequency - inverse document frequency) algorithm which transforms our features so as to improve as learners as described below.

4.2.1 TF-Transform

The TF (term frequency) transform is done to lower the impact common terms and document length on our weights. From our document-term matrix, which we denote as X, we apply the following transformation:

$$X_{ij} = ln(X_{ij} + 1)$$

where i refers to the ith document, and j is the jth term in that document.

4.2.2 IDF-Transform

The IDF (inverse document frequency) transform is done so that we increase the impact of rare words and lower that of common words. The transformation relies on the updated value of the matrix X from part 4.2.1. It is:

$$X_{ij} = X_{ij} * ln\frac{D}{N}$$

Where D is the number of training documents in our corpus and N refers to the number of documents in which word n occurs.

4.2.3 Length Normalization Transform

We then normalize each document in X in order to reduce the impact of longer documents with higher word counts. We apply the simple transformation on the matrix in part 4.2.2.

$$X_{ij} = \frac{X_{ij}}{\sqrt{(||X_i||)}}$$

Applying the above three transformations in the order of tf, idf, then normalization refers to the tf-idf transform.

5 Classifiers and models

We combined the data transformations as described in part 4 together with the following classifiers and models. This was in order to explore a mix of situations where we take into account the data biases as described in section 2.

5.1 Naive Bayes

We used a simple Naive Bayes classifier on our document-term matrix without applying any data transformations. The reason for this is because Naive Bayes decreases weights for classes with fewer training examples ie: not taking into account the skewed data bias. It also does not do any normalization for the feature weights. Thus this is an example of a poor text classifier for our data.

5.2 Transformed Weighted Complement Naive Bayes (TWCNB)

Complement Naive Bayes (CNB) is a model which aims to solve the problem of unbalanced data sets by looking at documents that belong to all classes except for the one we are examining. TWCNB is an extension of CNB where we use the tf-idf algorithm applied on our data set. Overall with TWCNB, we hope to solve the two biases that we explain in section 2.

To generate the class parameter vector for CNB, we take our tf-idf transformed matrix X and use: Let θ be an empty matrix of size C x N, where C is the number of classes, and N the number of words in our corpus.
For each point (c,n) in θ:

- For each training document k not in class C:
  - Let numerator be the sum of X_{kn}.
- Add one to the numerator.
- For each training document k not in class C:
  - For each word j in our set of unique words:
    - Let denominator be the sum of X_{kj}.
- Add the cardinality of the set of words to the denominator.
- Update point (c,n) to be numerator / denominator.

At this point, our class parameter vector takes into account the documents not in that class, but we now want to normalize. Define matrix W also with dimensions C x N. Let W be map(ln, θ) ie: taking the natural log of every point in θ and applying it to W.

We finally use an L1 normalization on each class c in W. This is because in our final step of classifying, differences in the magnitude of the weight vector may incorrectly cause our classification to prefer one class to another. Our classifier will then use the dot product between the word count vector of the test document, and the class parameter vector in W. The minimum result is the classification we want.

5.3 k-NN with SVD

k-Nearest Neighbor (k-NN) classification is a technique whereby the test vector is mapped onto the feature space of the training set, and classified based on a weighted vote of the points closest to it. In our paper, we applied SVD on our document-term matrix after it had been transformed with tf-idf. SVD used with text classification is known as Latent Semantic Analysis (LSA). The purpose of SVD is to reduce the number of dimensions in which we have train our k-NN classifiers while retaining important conceptual information. We transformed both the training and test sets, then used k-NN with a simple majority vote to classify documents.

5.4 Ensemble Classifier

We used a simple ensemble classifier that classifies documents based on the average count of the most common words. For example, the word “and” appears an average of 0.2 times per word for class A, but for class B it’s 0.1. For our test document, we find that “and” appears 0.14 times per word. Thus, it’s closer to B and based on this single classifier, we assign the document to B. We repeat this process with other common words and build up a simple majority vote classifier.

The average count reduces the impact of document length and unbalanced class bias, and so we also use it along with TWCNB as an example of a “good” classifier. The common words were selected based on the test document we wanted to process. The majority vote was weighted via distance, ie: in our “and” example, the document would have a distance of 0.04 to B, but 0.06 to A. So if we choose the 20 most common words, for example, we would sum up the distances of each individual classifier and our final classification would be the class with the lowest distance.

6 Implementation

All programs to run our algorithms were written in python. Raw data processing, removing “stop words” and the tf-idf algorithm were written from scratch, using the numpy python libraries for mathematical processing. We used the VarianceThreshold and TruncatedSVD modules from scikit to implement the Variance Threshold and SVD reductions.

For our classifiers, we implemented the TWCNB and Ensemble Classifiers from scratch while also using numpy for math processing. We used scikit libraries for k-NN and Naive Bayes classification.

7 Experiments and Results

We ran five different tests with the following classifiers: Naive Bayes, TWCNB, TWCNB with variance thresholding, k-NN with SVD, and our simple ensemble classifier on average word count data.

For Naive Bayes on our simple document-term matrix of word counts, we got the result that Hamilton was responsible for 10 of the 12 disputed papers, while Madison wrote the other two. For TWCNB, we got that Madison wrote 10 of the papers while Hamilton and Jay wrote one each. For TWNCB with variance thresholding, we got that Madison wrote 11 of the papers, while one was written by Jay. Finally, for our k-NN with SVD classifier with k = 1 (in essence, just a simple nearest neighbor), we got that half of the papers were written by Hamilton, and the other half by Madison. Figure 1 is a summary of the results.

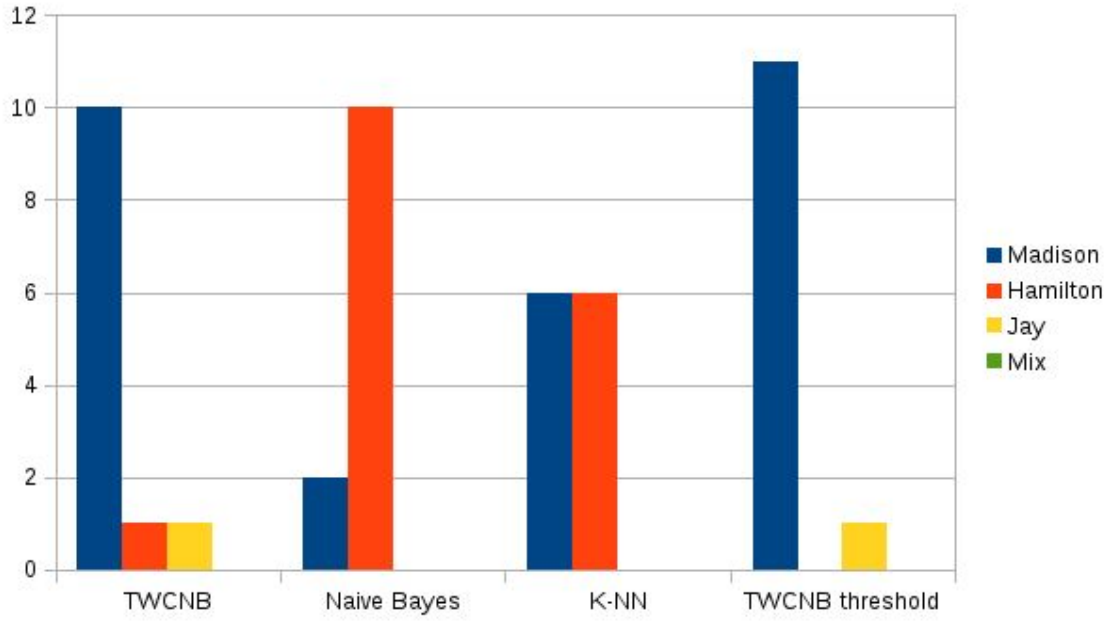

Figure 1.

7.1 Ensemble Classifier

For the ensemble classifier, we ran tests with the number of classifiers ranging from 1 to 15. When we add more feature to the ensemble, we are adding the classifier for the next most common word. We plotted in Figure 2 the effects of the ensemble classifier as we added more features.

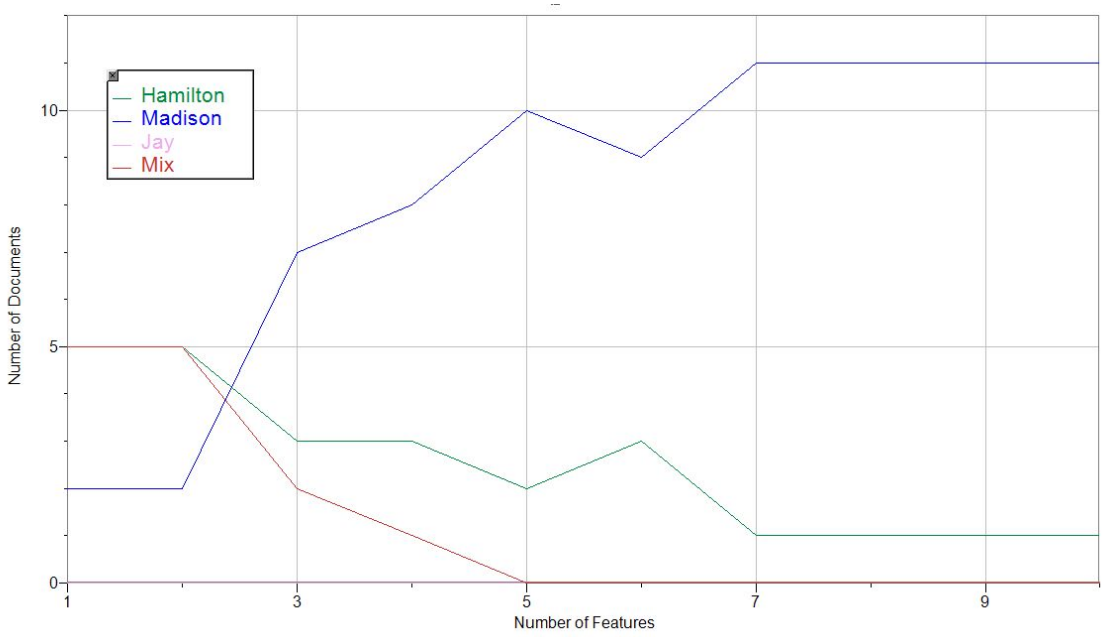

Figure 2.

We see from the graph that as we increased the number of features, the classifier began to classify more documents as Madison, until we reached a limit, where it would classify 11 of the documents as by Madison, and one by Hamilton.

7.2 k-NN

For figure 1, we ran our k-NN with SVD classifier with k = 1 and got that half of the papers were by Madison, and the other half by Hamilton. As we increased k, however, we started to get that Hamilton wrote more of the papers. The data is shown in figure 3.

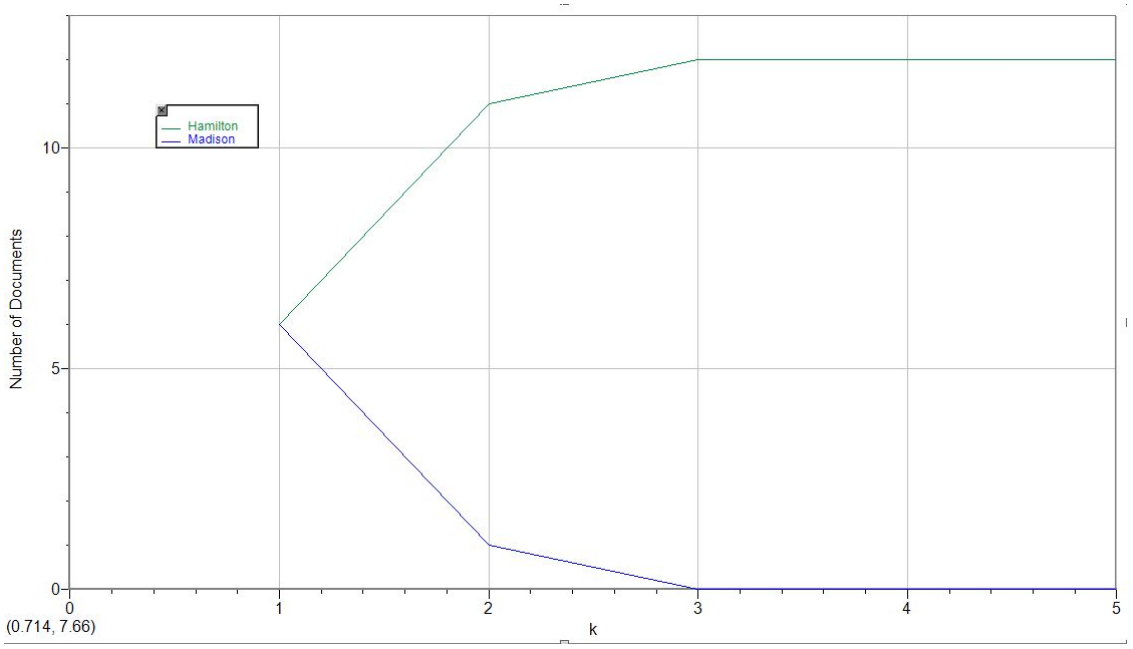

Figure 3.

8 Analysis and Conclusion

Based on historical consensus, we would like that we classify all 12 of the disputed papers as having been written by Madison. However, we are unable to do so, but we get some classifiers that classify the majority as having been written by Madison.

We see that the classifiers with the best results are the TWCNB, TWCNB with variance thresholding, and the Ensemble Classifier when we had at least 3 features. The worse performers are Naive Bayes and k-NN and Ensemble classifier with less than 3 features.

We can explain firstly why Naive Bayes and k-NN performed so poorly: Naive Bayes did not take into account the two biases we discussed and ended up predicting Hamilton the majority of the time since Hamilton was the majority class. k-NN performed decent when k was equal to one, but as we increased k, it also started to predict Hamilton. This is because with the majority vote scheme that we used it will always be skewed towards the majority class. So we can say that while k-NN took into account the bias of document length and unequal word counts, it did not take into account our skewed data.

TWCNB, TWCNB with variance thresholding and Ensemble Classifiers all performed well because these methods all take into account the two biases that we discussed. We saw that with the ensemble classifier, as we increased the number of features it started to perform better. This is because ensemble classifiers tend to perform better if we add more weak classifiers that perform better than chance. This can also explain the drop when we added the 6th feature - it was a classifier that did not perform better than chance.

Overall, we have shown that for a good text classifier it is necessary to take into account the data skew bias in your training set, and also to use appropriate data transformations that take into account document length and attempts to normalize them.

In the future, we would like perform all possible permutations of our classifiers and transformations, but we did not do so because of lack of time. Furthermore, we should probably have done some validation testing with our Ensemble Classifier - this way we could an algorithm like AdaBoost to determine weights to improve it, instead of our simple scheme whose weights were the distances between average word counts.

References

[1] Jason D. M. Rennie, Jason D. M, Jaime Teevan, and Jaime Teevan . Tackling the Poor Assumptions of Naive Bayes Text Classiffiers. Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.
[2] Guowei Zu, Wataru Ohyama, Tetsushi, Fumitaka Kimura. Accuracy Improvement of Automatic Text Classification Based on Feature Transformation. Faculty of Engineering, Mie University, Kamihama-cho, Tsu, Mie, Japan, Nov 2003.
[3] Sebastian Raschka. Naive Bayes and Text Classification I. arXiv:1410.5329v4 [cs.LG]. October 2014.
[4] Harshit D and Vikram P. Class Based Weighted K-Nearest Neighbor over Imbalance Dataset.Springer-Verlag Berlin Heidelberg, 305–316. 2013.
[5] Hiroshi S. Text Classification using Naive Bayes. February 2015.
[6] Padraig Cunningham. Dimension Reduction. Technical Report UCD-CSI-2007-7. August 2007.