Case Study 3: Spam Classifier

Using Term Frequency-Inverse Document Frequency Vectorization, K-Means Clustering, and Naive Bayes Classification

Kristin Henderson

June 7, 2025

1 Introduction

This study aims to predict whether an email is spam or not from a dataset of over 9,000 messages using Term Frequency-Inverse Document Frequency (TF-IDF) vectorization, clustering on the word count vectors, and a Naive Bayes classifier.

2 Data

The Spam Assassin dataset used in this study contains 9,353 emails consisting of three main types: plain text, HTML, and multipart messages. The messages contain a total vocabulary of 88,901 unique features. The target variable is binary, either spam or not spam.

Four messages were not able to be parsed into text with reasonable effort, so they were dropped from the analysis. One was plain text with an ASCII character set and the other three were multipart messages. Of the remaining 9,349 messages, 26% were spam and 74% were not spam.

The Python `email` package was used to parse the messages, determine their type, and extract the message content. BeautifulSoup was used to parse the HTML messages. Labels were obtained from the folder name of each message.

Prior to vectorization, the message text was cleaned by removing common control characters and most punctuation, leaving only letters, digits, underscores, apostrophes, and spaces. Extra whitespace was also removed.

To create a basic Bag-of-Words representation, the `CountVectorizer` was used to create a vocabulary of words across all messages and convert each message into a vector of word counts. In addition to the basic preprocessing listed above, the `CountVectorizer` lowercases text and tokenizes, extracting words made of two or more alphanumeric characters split at word boundaries.

3 Methods

3.1 Clustering on Basic Bag-of-Words Representation

Clustering was performed using two different algorithm-types, K-Means and DBSCAN.

K-Means was performed using both the default Euclidean distance metric and after applying L2 normalization. Since K-Means doesn't directly support cosine distance as a distance metric, L2 normalization was used because it preserves the vector direction but makes the length of each document vector equal to 1. This method emphasizes the similarity in the distribution of words rather than how many words were used, which helps reduce bias from document length.

The elbow method was used to determine the optimal number of clusters, within a range of 2-15, by identifying the point where the rate of decrease in inertia (within-cluster sum of squares) begins to level off. The silhouette score was also used to evaluate the quality of the clusters and help select the optimal number of clusters. The metrics of the K-Means clustering with L2 normalization are shown in Figure 1.

Using standard Euclidean distance, `k=9` was selected resulting in one large cluster of 8,497, two mid-sized clusters of 705 and 122, and six clusters of 10 messages or fewer. However, using L2 normalization, `k=10` was selected resulting in a much more balanced distribution of cluster sizes:

- Fewer than 500 messages: two clusters
- Between 500 and 1,000 messages: four clusters
- Between 1,000 and 2,000 messages: three clusters
- Over 2,000 messages: one cluster
- Minimum cluster size: 131
- Maximum cluster size: 2,735

To implement DBSCAN, similar to an elbow plot, a k-distance plot, shown in Figure 2, was created to approximate a reasonable `eps` value using the cosine distance metric. Then `eps` and `min_samples` were tuned in the ranges 0.4 to 1 and 3 to 15, respectively. The cosine distance metric was chosen because it measures the angle between document vectors, making it more suitable for text data than Euclidean distance which is sensitive to document length.

Using the cosine distance metric, the best `eps=0.9` and `min_samples=7` were selected by comparing total number of clusters, number of clusters greater than 10 messages, noise ratio and silhouette score. This resulted in 4 clusters, one very large cluster of 9,207, three small clusters under 30, a noise ratio of 1% and a silhouette score of 0.26, the highest of the parameter combinations tested.

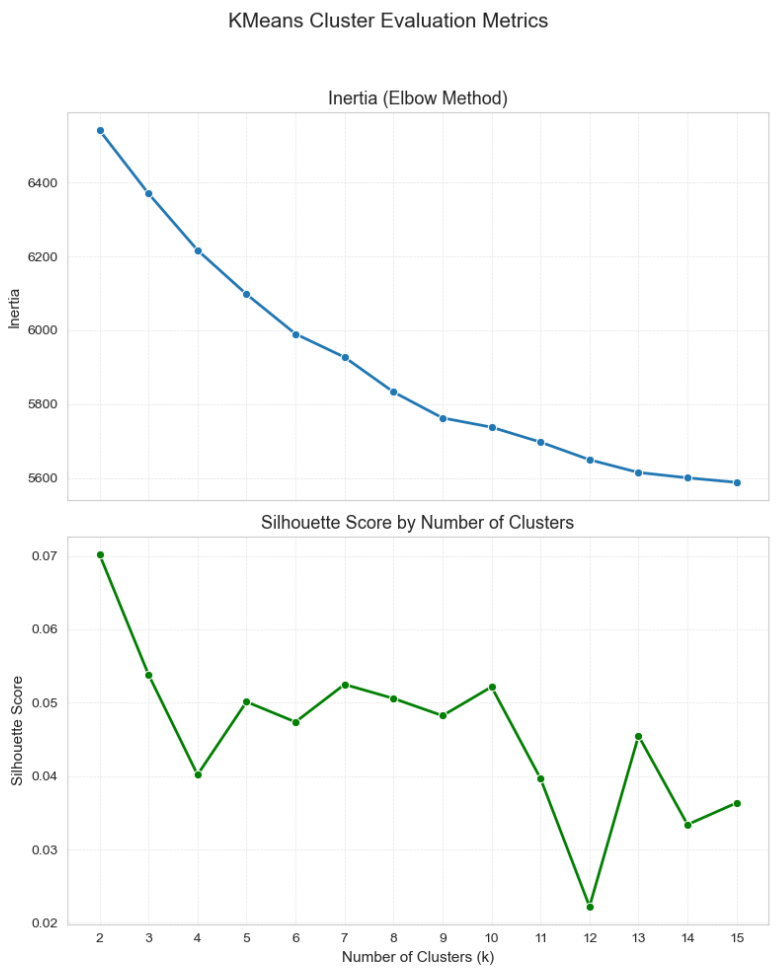

Figure 1: Elbow plot of inertia and number of clusters for L2 normalized count vectors (top) and silhouette score by cluster number (bottom). `k=10` was selected after considering both metrics.

3.2 Naive Bayes Classification on TF-IDF Representation

`TfidfVectorizer` was used to create count features for each word across all messages similar to `CountVectorizer`. Additionally, the `TfidfVectorizer` reweights the count features based on their frequency, resulting in floating point values that reflect each word's relative importance. TF-IDF represents each message as a vector of counts normalized for importance, a weighted Bag-of-Words representation.

After vectorization, separate datasets containing the TF-IDF features plus an additional feature containing cluster identity were created for each clustering

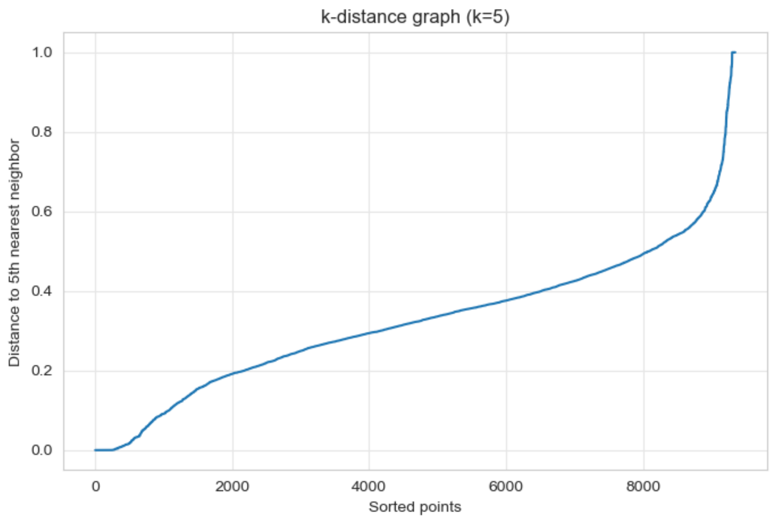

Figure 2: k-distance plot showing the distances to the 5th nearest neighbor for each message. A range of 0.4 to 1.0 was selected as a reasonable starting point for tuning `eps`.

method.

A manual grid search was performed on the TF-IDF/DBSCAN dataset using shuffled and stratified 5-fold cross-validation to optimize the smoothing parameter, `alpha` in the range of 0.1 to 1.0. This was done for both a Multinomial and a Categorical Naive Bayes classifier. For the Categorical Naive Bayes model only, the features were discretized into 5 bins using the `KBinsDiscretizer` with an ordinal encoding since this model requires discrete features. The same was repeated for the TF-IDF/K-Means (Euclidean) dataset. All metrics evaluated, accuracy, F1 score, precision and recall, were highest at an `alpha` of 0.1.

A similar cross-validated grid search was performed on the TF-IDF/K-Means (L2 normalized) dataset with a lower range of `alpha` values from 0.01 to 0.5. Again, all metrics were highest at the lowest `alpha` tested, this time at 0.01.

The grid search was also performed on the TF-IDF features without cluster identities to compare performance and confirm that including cluster identities improved the scores. Additionally, a model was created using one-hot encoded K-Means cluster features to test if removing the implied ordinal structure would improve performance. This confirmed that adding a single cluster feature was sufficient and one-hot encoding wasn't necessary.

4 Results

The Multinomial Naive Bayes model outperformed the Categorical Naive Bayes model across all datasets and metrics, except for precision. The dataset with

L2 normalized K-Means cluster identities achieved higher scores than the other datasets across all metrics except precision. The dataset without cluster identities achieved a higher precision, though the difference was minimal.

The final model was fit on the TF-IDF/K-Means (L2 normalized) dataset with an `alpha` of 0.01 and the Multinomial Naive Bayes classifier. To visualize the cluster separation, PCA was performed on the L2 normalized count vectors (not the TF-IDF features) and the first two principal components were plotted with cluster identities indicated by color, shown in Figure 3.

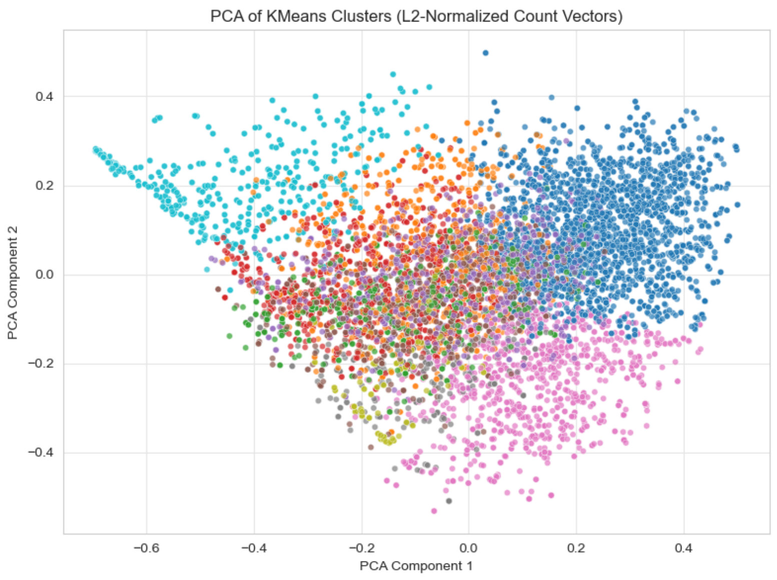

Figure 3: PCA visualization of L2 normalized count vectors (not TF-IDF features) using the first two principal components. Colors indicate K-Means cluster assignments.

The overall accuracy was 0.9919 (Table 1), with a balanced accuracy, the average true positive rate of both classes, being a bit lower at 0.9867. Weighted precision, recall and F1-scores were 0.9919, 0.9919, and 0.9918 respectively. Because these metrics were so high, it wasn't necessary to select a classification threshold other than the default of 0.5, despite the classes being imbalanced.

Due to the class imbalance (26% spam, 74% non-spam), balanced accuracy and weighted metrics were used to evaluate performance across both classes, balanced accuracy averages the recall scores of both classes, giving equal weight to each regardless of the class size or frequency, and weighted metrics weight the class-specific metrics by their frequencies to account for the class imbalance.

Each class performed with similarly high metrics. Precision, recall and F1-scores, shown in Table 2, were 0.9920, 0.9762, and 0.9840 for the spam class and 0.9919, 0.9973, and 0.9946 for the non-spam class.

Table 1 contains the overall metrics from the cross-validated model with the default `argmax` threshold.

Table 1: Overall Cross-Validated Out-Of-Fold Metrics

Metric	**Mean**	**Standard Deviation**
Accuracy	0.9919	0.0024
Balanced Accuracy	0.9867	0.0040
Weighted F1-score	0.9918	0.0024
Weighted Recall	0.9919	0.0024
Weighted Precision	0.9919	0.0024
ROC AUC	0.9996	0.0002

Table 2 contains class-specific results from the classification report with the default `argmax` threshold.

Table 2: Metrics by Class

Metric	**Spam**	**Not Spam**
F1-score	0.9840	0.9946
Recall	0.9762	0.9973
Precision	0.9920	0.9919

The confusion matrix is shown in Figure 4. The model correctly classified 2,338 of 2,395 spam messages as spam and 6,935 of 6,954 non-spam messages as non-spam. It performed slightly better on the non-spam class than the spam class, reflected in the higher recall for the non-spam class (0.9973 ± 0.0015) compared to the spam class (0.9762 ± 0.0075).

The ROC Curve on out-of-fold predictions is shown in Figure 5. The area under the curve (AUC) was 0.9996 ± 0.0002.

Figure 6 shows how different words are associated with spam versus non-spam messages. Each bar represents how much more likely (positive values) or less likely (negative values) a word is to appear in spam compared to non-spam messages. These are log probabilities of word frequencies, where each bar represents the logarithm of the likelihood ratio between spam and non-spam messages. The top 10 words that best distinguish between spam and non-spam are shown.

5 Conclusion

Using TF-IDF vectorization, K-Means clustering with L2 normalization, and a Multinomial Naive Bayes classifier with an `alpha` of 0.01, the cross-validated model achieved an overall accuracy 99.2%. Its balanced accuracy was only slightly lower at 98.7%, because of the imbalanced classes and lower performance of the classifier for the spam class. Spam was slightly more difficult to classify, with a recall score of 97.6% versus the 99.7% recall of the non-spam class.

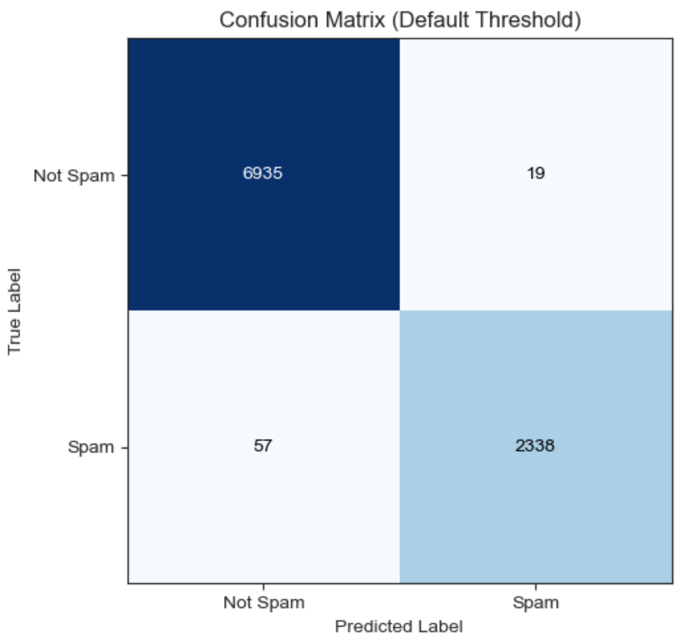

Figure 4: Confusion matrix with the default (`argmax`) threshold.

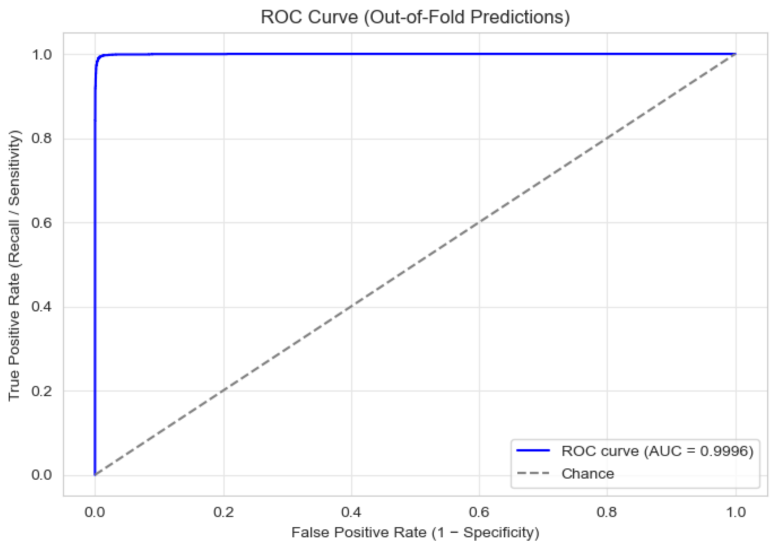

Figure 5: ROC Curve of true positive rate versus false positive rate on the out-of-fold predictions.

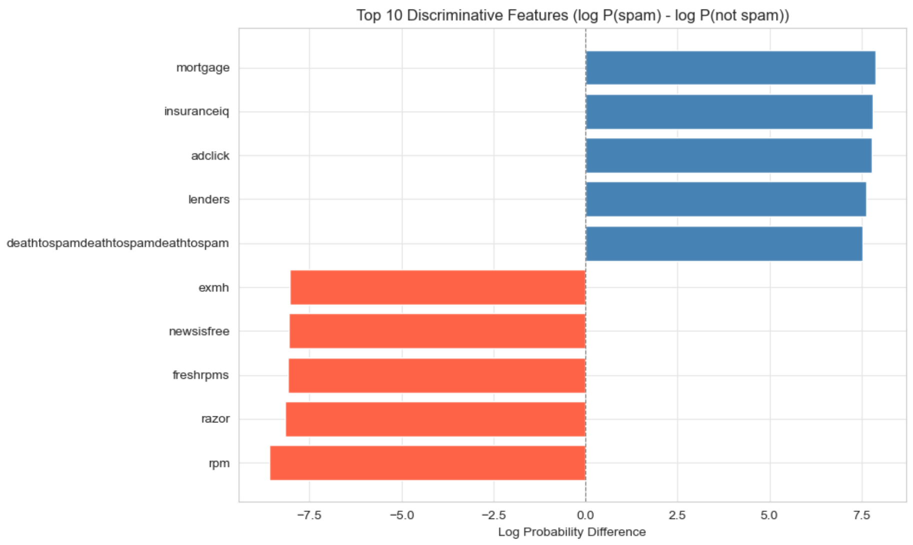

Figure 6: Top ten most discriminating features and their log likelihood ratios between spam and non-spam classes.

Depending on the situation and preferences, it may be more advantageous to favor detection of non-spam to spam to prevent important messages from being filtered to a spam box and potentially missed by a user. A savvy user can filter the 2.4% of spam messages missed by the model, which is arguably easier than regularly checking a spam box.

Interestingly, words like mortgage and lenders are more likely to appear in spam while razor is more likely to appear in non-spam. Additionally, the clustering appeared to improve the model's performance, though it's unclear which clusters were stronger indicators of spam and which of non-spam. To gain further insight into the predictive power of the cluster labels, it may be beneficial to use one-hot encoded cluster features instead of the single cluster feature to better understand how each cluster contributes to classification.