**Project 2 SVM Feature Extraction/Selection Comparison**

Chase Lamkin 23 April 2023

## Project Description

This project aims to use a support vector machine (SVM) in conjunction with different feature selection and extraction methods, to detect spam emails, which is an ongoing and increasing problem across multiple platforms. The spam email dataset used is Kaggle's public spam detection dataset. The project asks the user which extraction and selection methods to use and then displays the accuracy results the of the specified approach. This allows for comparisons of time and accuracy of different approaches and allows users to make the decision for themselves which method would be suitable for their own spam detection.

## Program Description

The goal when implementing this program was to avoid the use of high level python libraries in order to give fair comparisons. Due to this the SVM used is a self-implemented version which minimizes the cost function over 2000 iterations. The feature extraction methods are Bag of Words (BoW) and Term Frequency - Inverse Document Frequency (TF-IDF). These methods are also self-implemented, which allowed me to use the same regex extraction on both. It also allowed me to manipulate the pandas data frame as needed. The feature extraction methods, however, used high-level python libraries. These methods include variance threshold seletion, chi-squared test selection, and principal component analysis (PCA) selection. The reason behind this, is that without the use of a neural network to generate word embeddings, getting the number of features below 7000 is near impossible. With such a high number of features self implemented methods are likely to take too long (an attempt at a self-implemented correlation method rendered the terminal unresponsive). Finally, the SVM uses exactly one feature extraction technique and zero or one feature selection techniques when determining accuracy.

**Implementations**

- **SVM -** The SVM implementation first reads the dataset csv file into a pandas data frame and then splits the dataset into the two data frames X and Y. Y is the target (ham or spam) while X is the email text (features data frame). Extraction is then performed, using the specified methods I will go over later, on the email text. Then the Y data frame is converted into values (ham = 1, spam = -1) to allow the SVM to quantify predictions. The features data frame is then parsed into the specified feature selection method, I will also specify later, and returns a data frame, which preferably has less then the original number of features (7000+). The features data frame is then normalized, to allow for faster compute times, and a last column of ones is added to the features dataframe to account for the b intercept of the hyperplane. Finally, both the target and features data frames are split into train and test (80% = train, 20% = test). The weights are then passed into the stochastic gradient descent function which calculates the gradient and minimizes the cost over 2000 iterations. I used a low number of iterations and did not use stoppage criteria for this function to allow for fair comparisons of different feature selection and extraction methods and to also prevent long load times caused by 7000+ features. Finally the weights are calculated and the predictions are tested against the prediction data frames. Results are then displayed

- **Bag of Words** - For the BoW feature extraction I fit the model by creating a total vocabulary set using parsing techniques on each document in the data frame. First I added all special characters to the set. This is important because spam messages typically contain a high number of them, so they needed to be accounted for like words. Its also important that the special characters are also all accounted for separately as not all special characters hold the same weight for predictions. Next regex is used to to find all numbers and words interlaced with numbers. These are also all added to the vocabulary set, however, all numbers are accounted for as the word “numberVal” and all words that are interlaced are accounted for as the word “wordNumber.” This is important because it drastically reduces the number of features (by over 500+) and because almost all numbers and interlaced numbers are unique despite holding the same importance. Finally the documents are split into words and added to the set. After the model is fit I transform it by parsing each document individually, and

adding the frequency of each word used into a dictionary for each document. These dictionaries are then appended to a master list, which is used to create columns that replace the “EmailText” column. Each of the new columns represents one feature, which is now quantified. This implementation of feature extraction saves on time and allows for emphasis on important words, but its weakness is that it does not account for word placement.

- **TF-IDF** - The TF-IDF function used in my implementation is: d_freq(term) * log((1+D) / (1 + freq(term))), where D is a datapoint (document) in the “EmailText” column, d_freq(term) is the frequency of a term in a document, and freq(term) is the number of times a term is found across the entire dataset. Using this formula my TF-IDF object transforms the fitted dataset, document by document, into dictionaries which are then added as rows into a new data frame. Most of the implementation of fitting this dataset is similar to the BoW implementation, however, notable differences is that the TF-IDF implementation also keeps track of the overall frequency of all terms in the vocabulary when fitting and also keeps track of the frequency of terms in each document when transforming. My implementation does this through the use of a default dictionary (defaults to 0), which allows me to increment terms before I know the entire vocabulary. This also allows me to prevent redundantly parsing the dataset, which is expensive.

- **Feature Selection** - The implementation of feature selection methods was through the sklearn library to allow for consistent implementation across all methods. The first selection method, uses Sklearn’s VarianceThreshold to remove redundant features below the variance threshold, which is user specified. The second selection method uses Sklearn’s SlectKBest class to run a chi-squared test on the data frame to determine the affect of each feature on targets through statistical analysis. The user specified number of best features are then kept in the training and testing data frames. The last selection method uses Sklearn’s PCA class to perform principal component analysis on the data frame reduce the dimensions (number of features) to a user specified number. The goal of this is retain the maximum amount of information possible by grouping the vectors. It's important to note that the functions I provided to implement these classes only take in a pandas data frame and the results of the classes are

modified to return a pandas data frame (with modified column names in the case of PCA). This allows implementation of these classes with my SVM.

**Results**

- **SVM -** There are no results to compare the SVM to in this project, but it is important to note that this programs' implementation performed exceptionally well, never going below a 95% accuracy rate with any of the selection methods, except variance selection.
- **BoW** - The BoW implementation achieved a 98.47% accuracy rate (without feature selection) on average. However, this astoundingly high accuracy was overshadowed by the, on average, 8-10 minute training time. This is due to the high dimensionality of the features.
- **TF-IDF** - The TF-IDF implementation achieved a marginally better, 98.75% accuracy rate (without feature selection) on average. Surprisingly, this accuracy rating is **even higher** than the accuracy I was able to achieve using Sklearn's own TF-IDF class. This however, is overshadowed again by the 8-10 minute training time due to high feature dimensions. The total time actually went up on average using this method by 30 seconds - 45 seconds, due to increased extraction time.
- **Variance Selection** - Feature selection using variance performed astronomically bad due to the high number of dimensions of the dataset. On average, any variance selection over 40% caused the accuracy of the SVM to fall below 95%. This is because due to the high dimensionality of features, many documents contain thousands of words with zero frequency. This causes variance between features to fall below the typical 98% selection rate.
- **K-Best (chi squared)** - Feature selection using Chi-Squared performed on average, at a 97.5% accuracy when selecting 35 to 45 features. While, this did perform worse than the BoW and TF-IDF extraction methods without selection, it was only a .99% relative decrease. More importantly the training of the K-Best method was on average 4-5 minutes, an 50% decrease.
- **PCA** - Feature selection using Principal Component Analysis, performed on average with a 98.4% accuracy when using 450-550 features. The training time

decrease on average was 2-3 minutes. This is an impressive decrease in time while maintaining, accuracy within .07%.

## Conclusion

Both BoW and and TF-IDF extraction techniques were effective with minimal time and accuracy differences. TF-IDF performed only marginally better, with a relative .2% increase in accuracy, which is not enough to account for the average 30 second increase in time. The lack of increase in accuracy is likely due to not accounting for word positions as well as an already high accuracy achieved by the BoW extraction. Decreasing feature dimensions using the Chi Squared test performed the best in comparison to its increase in time efficiency. This project shows that with the options offered, feature selection of text with a large number of variables such as spam, is most resource and time efficient using a combination of BoW for feature extraction and Chi Squared feature selection.

## References

https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

https://users.cs.utah.edu/~zhe/pdf/lec-19-2-svm-sgd-upload.pdf

https://en.wikipedia.org/wiki/Support_vector_machine#:~:text=More%20formally%2C%20a%20support%20vector,other%20tasks%20like%20outliers%20detection.