Kwasi Mensah

IST 664 Natural Language Processing

NLP Final Project

Detection of SPAM in Email

Introduction:

For our analysis we will try to detect Spam in emails with text files that were collected. The emails obtained to create our corpus will be from the Enron public email corpus. In 2001 a lawsuit against the energy company Enron made a large number of their company emails public which led to the creation of this corpus. On top of the Spam emails already in the corpus, more spam emails are added in order to have a sample large enough for analysis and training and classifiers that are created. The text files collected mirror the first three directories of both Spam and Ham emails. This set contains 3672 regular email text files that are in the ham folder and 1500 Spam emails in the spam folder. We will perform a sentiment and subjectivity analysis at first to try and see if we could distinguish between spam and ham emails this way. In addition, we will train and test different types of classifiers to see how accurate our models will be at predicting spam and ham emails respectively. Lastly, we will test our classifiers when new data is introduced, which will be spam and ham emails from the “Enron2” data set.

Method:

We first import the ‘os’ library in python, which is a library used to interact with our operating system. We use this in order to parse through the folders with each respective email. We also import the ‘chardet’ library is used for detecting the encoding of the text files in the folder. We create nested for loops in our python code in order to parse through the folder downloaded that contains the email files. The loop will go through each file that has the ‘.txt’ extension and save them into a new object called ‘file\_texts’. Next we store each folder name as the label for each email text file that were collected from the respective ‘ham’ and ‘spam’ folders in a new object called ‘labels’. We next print the number of labels and text files collected to check to see that we have an equal number of observations. Next we import the pandas library in python in order to create a data frame from the two lists created and name the data frame ‘spamdf’. We next check to see if the data frame was correctly created by looking at the first 5 rows of our data frame. Lastly we check to see the count of both ham and spam emails in order to ensure that we have 3672 ham emails, and 1500 spam emails. In addition, we create a visualization of a bar plot to visualize these counts.

Once this is confirmed, we next perform an exploratory analysis of the data collected that will eventually become our corpus. We first check to see what the average character count of spam and ham emails is. We do this in order to see if there may be a way to try to differentiate between the two types of emails. Once this is done we next perform a sentiment analysis on our data frame. To begin with we remove any duplicates that may appear in our data frame. Next we import the ‘Textblob’ library and create two different functions one to get the polarity score which we will use to determine the sentiment of the emails, and another function to determine the subjectivity of the emails being analyzed. Any polarity scores less than 0 will be deemed negative, greater than 0 as positive, and anything in between as neutral. When it comes to subjectivity any scores less than .5 is negative, greater than .5 is positive, and equal to .5 as neutral. We will compare the count of each respective group to see if this would also be a good determining factor to distinguish between ham and spam emails.

We now begin the tokenization of emails to create a corpus, but before doing so we remove the word ‘Subject:’ from all of our observations. We do this by using the string replace function provided by python to remove this string of characters. We next import the NLTK library as well as the associated packages like ‘word\_tokenize’, ‘PorterStemmer’, ‘stopwords’ and set the stop words to be the English library. We also import the regular expression library to remove any URLs or web addresses, any non-space characters with empty strings, and remove any numbers that may appear in any of the emails. We use all of this to create a function that will perform all of these preprocessing sets and a separate function to get the text into their stem form using our Porter Stemmer. Once this is completed we check to see our data frame with our tokenized corpus and again drop any duplicates that may have been created during our preprocessing steps.

We balanced the dataset in order to make our training model perform better and make better predictions. We do this in order to avoid our model from being biased when classifying between given trained classes. In addition, doing this we can try to avoid over fitting our classification models, because we have a larger number of ham emails when compared to spam emails. We do this to avoid our models favoring the majority and give equal priority to each of our available labels. If not, we could run into our classifier making more predictions in favor of ham emails. We then down sample the number of ‘ham’ messages are reduced to match the number of spam email messages. In addition, for comparison we up sample the number of “spam” email messages meaning we increased the number of samples to match the number of ‘ham’ messages. Both down sampled and up sampled ham and spam messages were concatenated respectively and create a bar plot to confirm visually that we have an equal number of observations for both sets for us to begin our analysis. We also create a word cloud to visualize the tokens that we will be analyzing in our corpus. We also create an additional word cloud with added stop words such as, ‘email’, ‘spam’, ‘message’, ‘click’, ’link’, 'http', 'www', 'hou', 'com', 'cc', 'etc', 'bcc', to get a clearer word cloud with less clutter.

To create our model, we first created three different vectorizers to use with our classifiers. We created different vectorizers with different features, such as boolean vectorizer, unigram and bigram term frequency vectorizer, and a unigram TF-IDF vectorizer, all with a minimum frequency count of 5. We loaded the ‘CountVectorizer’ and ‘TfidfVectorizer’ through the scikit-learn library in python. We next separate the values in the text and label columns from the data frame into separate variables and split the ‘X’ &’y’ variables created into training and testing sets. We do so by using the ‘train\_test\_split’ function provided for us by the scikit-learn library and perform a 70% to 30% split of the variables. We then fit vocabulary in our corpus and transform it into vectors from the training and testing sets created using each respective vectorizers created. We do this both for our up sampled and down sampled data sets to compare the accuracy of any classifier created.

We next import our Multinomial Naïve Bayes module through ‘sklearn’ and initialize the MNB model into a callable variable. We then use the training data to train our MNB model and test the classifier on the test data set. We then print out accuracy score of the models created from each down sampled and up sampled data set, using each of the three vectorization options. Once we receive our result for our MNB models we now will create our models for our Support Vector Machine using the same vectorization options. We import the “LinearSVC” module from sklearn and initialize the LinearSVC model in a variable. Then we call on this variable to use the training data to fit and train the model. We do not transform the data set for our SVM models but we do look for the accuracy scores of the models created and compare them to our previous results. We next look at the classification report to examine the precision which measure the number of true positive predictions that were correctly made, recall which tells us the percentage of instances that belonged to their respective classes, and F1-Scores which is the weighted average of both precision and recall score. With the F1-Score we can see if there is a good balance between the recall and precision scores of a model. We also imported the GridSearchCV module from ‘sklearn’ for us to try to find the best parameters and configurations we manually set for our SVM models. We use this to find the best hyperparameters for each of the three vectorization techniques used on the respective models.

Lastly, to test how well the classifiers that we created to distinguish between spam and ham emails we decided to introduce new data and see if it will predict the same labels for each email introduced. We use the Enron-Spam pre-processed data Enron2 to test the models with the best scores. We will use similar functions as we previously created to parse through the folders and create separate data frames for the new test set. The ‘label’ columns are first duplicated and named ‘true\_label’ and clear the contents of the original column duplicated. This will be the column we use our classifiers to make predictions on the correct label. We use the same preprocessing techniques to clean and tokenize the emails in this test set. We then check the count out the spam and ham emails in this test set and begin to test both our MNB and SVM classifiers to see which one has the best results in predicting spam and ham emails.

Results:

First we look at the results of our exploratory analysis using the Textblob library in python and the average character count. When we look at the average character count we see that for ham email messages the average email contains around 959 characters, and for spam email they contain on average about 1204 characters. We can see by looking at these character counts that spam emails on average are longer than normal emails, which is to be expected as they are usually advertisement emails or some sort. When we look at our sentiment analysis of our email corpus we see for ham emails that 672 emails have negative sentiments, 780 neutral emails, and 1975 positive emails. When we look at the spam emails’ sentiment we see 145 negative emails, 177 neutral emails, and 1137 emails that were classified to have positive sentiment. If we investigate how subjective the emails are for each respective label we get 31 neutral ham emails, 2899 objective emails, and 497 emails as subjective for ham emails. When we look into the spam emails subjectivity, we see 28 neutral emails, 1012 objective emails, and 419 subjective spam emails. We can see that by looking at both ham and spam emails subjectivity and sentiment it would be difficult to find and difference that would be distinguishable as they each have similar weight for each category. We see that after preprocessing, removing duplicates, and tokenization, we see we have 1457 spam emails remaining and 3174 ham emails remaining for analysis. We will use these counts to create the up sample and down sample data sets.

When we look at our six Multinomial Naïve Bayes Classifiers that were created using each of the 3 vectorization objects we created earlier. We ran our MNB classifier on both the concatenated down sampled and concatenated up sampled messages. When we use the boolean vectorizer we get an accuracy score of about 96.69% for MNB classifier that uses the down sampled set, and an accuracy of about 97.57% when using the up sampled set. When we use our bigram term frequency vectorizer for our down sampled set we get an accuracy score of 97.15% and for our up sampled set we get a score of 98.06%. Lastly, when we test our MNB classifier created with our unigram TF-IDF vectorizer, we get an accuracy score of 96.81% for our down sampled set and 97.42% for our up sampled set. We can see from these results that our up sampled data set yields expected higher accuracy for our classifiers. Next we print out the classification report to see the precision, recall, and F1-Scores for both SVC that had the best accuracy scores.

Next we print out the classification report to see the precision, recall, and F1-Scores for both MNB models that had the best accuracy scores, which were the MNB model created with the bigram term frequency vectorizer and the boolean vectorizer, but both used the up sampled corpus. We see for the precision scores bigram term frequency vectorizer we get a score of 98% for ham and 97% for spam, for recall scores we get 97% for ham and 98% for spam, and for the F1-Scores we get 98% for both spam and ham. In terms of the MNB model that used the boolean vectorizer for the precision scores we get 99% for ham and 97% for spam, for recall scores we get 97% for ham and 99% for spam, and for the F1-Scores we get 98% for both spam and ham.

We train a total of six Linear Support Vector Classifiers that use different vectorization techniques and hyperparameters. We use the same 3 vectorizers as our MNB model unigram to compare the results. We ran our Linear SVM classifier on both the concatenated down sampled and concatenated up sampled messages. When we use the boolean vectorizer we get an accuracy score of about 97.26% for SVM that uses the down sampled set, and an accuracy of about 98.69% when using the up sampled set. When we use our bigram term frequency vectorizer for our down sampled set we get an accuracy score of 97.03% and for our up sampled set we get a score of 98.44%. Lastly, when we test our SVM created with our unigram TF-IDF vectorizer, we get an accuracy score of 98.74% for our down sampled set and 99.17% for our up sampled set. We can see from these results that our up sampled data set yields expected higher accuracy for our classifiers.

Next we print out the classification report to see the precision, recall, and F1-Scores for both SVC that had the best accuracy scores, which were both the SVM classifiers created with our TF-IDF vectorizer. We get the same results for both the up sampled and down sampled data sets in terms of precision, recall, and F1-Scores. We see for the precision scores we get 100% for ham and 98% for spam, for recall scores we get 98% for ham and 100% for spam, and for the F1-Scores we get 99% for both spam and ham. Next we will perform a grid search to help us find the best parameters for our SVM classifier. For our grid search we set the search regularization parameters(‘C’) to .01, .1, 1, & 10 to see which degree of flexibility our models need to be when accepting errors, and perform a 10-fold cross validation when performing these searches. Unfortunately, we did not get higher accuracy scores than the ones we got in the grid search which suggested we used .1 for majority of our vectorizers but did have the same ‘C’ of 1 for the highest resulting accuracy as our original models that uses the unigram TF-IDF vectorizer did. Next we will use the best two SVM models and compare them to our two best MNB models when it comes to predicting labels when new data is introduced.

When checked our cleaned test corpus we now have 4138 ham messages and 1491 spam messages, after following the same preprocessing steps as we did for our training corpus. When we use the SVM model named svm\_clf5 we get 3725 messages labeled as ham and 1904 messages labeled as spam. This specific model was 90.993% accurate in identifying spam and ham emails. When we use the SVM model named svm\_clf4 we get 3542 messages labeled as ham and 2087 messages labeled as spam. This model gave us 87.671% accuracy at making predictions of the two labels. Next we will look at how accurate our MNB models are at distinguishing new data that has spam and ham emails. To compare, when we use the MNB model with the highest accuracy score named nb\_clf we get 3719 messages labeled as ham and 1910 messages labeled as spam. For this classifier it is about 87.635% accurate when new data is introduced. Lastly, when we use the MNB model named nb\_clf3 we get 3727 messages labeled as ham and 1902 messages labeled as spam. This classifier actually produces the most accurate prediction of spam and ham emails with an accuracy of about 91.491%,

Conclusion:

In conclusion we can see that looking at the average character count of emails is a good initial start to finding out ways to distinguish between spam and ham emails. When we try to find differences in types of emails when looking at their overall sentiment and how subjective the emails are, we see that this is not the most feasible way in order to try to find any difference between spam and ham emails. We also see that when comparing the accuracy when using up sampled and down sampled data sets, up sampled sets will yield higher accuracy most likely due to having a larger number of samples to base its predictions from. We can conclude that since the new data that was introduced to the models had more entries when compared to the original training set it is understandable to see our predictor not having 99% accuracy as expected.

It was interesting to see that the classifiers with high accuracy scores classified a significant number of ham emails as spam emails. This resulted in having a larger number of spam emails than expected. When using the GridSearchCV library provided by ’sklearn’ is a great tool to use to double check if we have created the best model possible without overfitting. In addition, it could be a great tool to confirm if working alone, that the model to be tested accuracy score is the highest we could possibly get with the corpus being analyzed. We would also want to look into ways to perform the same search for possible hyperparameters to merge when developing Multinomial Naïve Bayes classifiers.

It is also interesting to note that our MNB classifier classified 2 more ham messages correctly compared to our SVM model that had the best overall accuracy score, thus being about 1% more accurate in comparison. Finding this result opened the door for more future analysis as to why the MNB model has a lower accuracy score but performs better at making predictions than our SVM which hyperparameters were found for. In the future, we may want to try this experiment again with our classification models, but by manually combining an equal number of “spam” and “ham” messages to see if we get similar accuracy scores. Additionally, we would want to see if we could get better results when creating these classifiers for testing to produce a high-performance model.

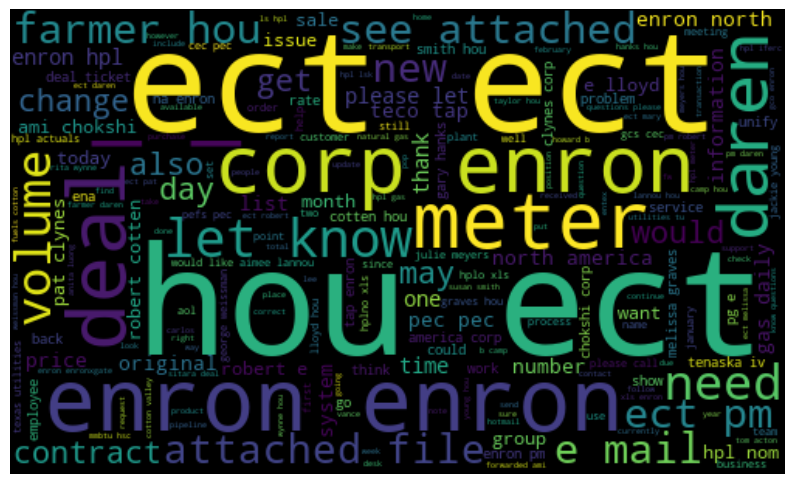
WordClouds:

Spam

Text

Description automatically generated

Ham



References:

1. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825-2830. http://jmlr.csail.mit.edu/papers/volume12/pedregosa11a/pedregosa11a.pdf
2. Joachims, T. (1998). Text categorization with Support Vector Machines: Learning with many relevant features. In Machine learning: ECML-98 (pp. 137-142). Springer. https://www.cs.cornell.edu/people/tj/publications/joachims\_98a.pdf
3. DataCamp. (n.d.). WordCloud Python Tutorial. Retrieved from https://www.datacamp.com/community/tutorials/wordcloud-python.
4. Topcoder. (2018, May 24). Getting Started with TextBlob for Sentiment Analysis. Retrieved from https://www.topcoder.com/thrive/articles/getting-started-with-textblob-for-sentiment-analysis#:~:text=When%20a%20sentence%20is%20passed,to%20personal%20opinions%20and%20judgments.
5. Garg, N. (2020, July 21). 10 Techniques to deal with class imbalance in machine learning. Retrieved from https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/.
6. Pedregosa, F., & Scikit-learn contributors. (2021). train\_test\_split - scikit-learn 1.0 documentation. Scikit-learn. Retrieved March 30, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.train\_test\_split.html.
7. Scikit-learn. (n.d.). sklearn.svm. LinearSVC. Retrieved from http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html.
8. Scikit-learn. (n.d.). sklearn. naive\_bayes.BernoulliNB. Retrieved from http://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.BernoulliNB.html.