Spam Classification

Building a Spam Classifier from scratch involves majorly two main tasks:

1. Choosing/ Building a reliable dataset. The texts in the emails should span the set of all words belonging to the spam distribution/ non spam distribution for Spam and Ham emails respectively.
2. Choosing a Binary Classification model{ e.g., Logistic Regression, Naïve Bayes Classifer} for the task.

Dataset:

For Spam Classification task, two datasets Enron Spam and Ling-Spam are merged to form one single dataset. These datasets are obtained from Natural Language Processing Group Department of Informatics - Athens University of Economics and Business.

Enrom Spam dataset is featured in the paper " *V. Metsis, I. Androutsopoulos and G. Paliouras, "Spam Filtering with Naive Bayes - Which Naive Bayes?". Proceedings of the 3rd Conference on Email and Anti-Spam (CEAS 2006), Mountain View, CA, USA, 2006*".

Ling-Spam dataset is featured in the paper "*I. Androutsopoulos, J. Koutsias, K.V. Chandrinos, George Paliouras, and C.D. Spyropoulos, "An Evaluation of Naive Bayesian Anti-Spam Filtering". In Potamias, G., Moustakis, V. and van Someren, M. (Eds.), Proceedings of the Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning (ECML 2000), Barcelona, Spain, pp. 9-17, 2000*".

In both the datasets each message is in a separate text file To prepare the merged dataset, the files were segregated into two folders: Spam and Ham. In Enron Spam dataset all the spam messages have the word "spam" in the filename and in Ling-Spam files whose names have the form spmsg*.txt are spam messages. These spam messages were moved into the Spam folder, while non-spam messages were placed in the Ham folder.

The merged dataset contains a total of **9,495 spam messages** and **14,650 ham messages**.

Classifier:

The Logistic Regression model is built from scratch to perform the classification task.

The data undergoes several preprocessing steps before training the model. Below is an example of a sample email before applying the preprocessing steps:

'Subject: fw : this is the solution i mentioned lsc\noo\nthank you ,\nyour email address was obtained from a purchased list ,\nreference # 2020 mid = 3300 . if you wish to unsubscribe\nfrom this list , please click here and enter\nyour name into the remove box . if you have previously unsubscribed\nand are still receiving this message , you may email our abuse\ncontrol center , or call 1 - 888 - 763 - 2497 , or write us at : nospam ,\n6484 coral way , miami , fl , 33155 " . 2002\nweb credit inc . all rights reserved .'

The preprocessing steps involve:

1. Converting all the characters to lower case.
2. Removing number, punctuations and special characters from the sentences.
3. The sentences is stripped off the new line characters.
4. The above operations leads to extra spaces between the words in the sentences, these extra spaces are removed.
5. Lemmatization is applied which reduces words to their base form while preserving their meaning and context.
6. Removing stop words to eliminate common words that do not contribute significant meaning.

The above email, after applying the preprocessing steps, results in the following:

'subject fw solution mention lsc oo thank email address obtain purchase list reference mid wish unsubscribe list please click enter name remove box previously unsubscribed still receive message may email abuse control center call write us nospam coral way miami fl web credit inc right reserve'

The **Count Vectorizer** is used to convert text into numerical form by creating a bag-of-words model. In this model, each word is treated as a separate feature, and the count of each word in a given document serves as the value for that feature. This process results in a sparse matrix where each row corresponds to a document and each column represents a unique word from the corpus. The values in this matrix indicate the frequency of each word in the respective documents.

The **min_df** parameter of count vectorizer is set to “0.005” such that the fit_tranform method learns those unique words from the X_train data which appear in atleast 100 documents [0.005 x len(X_train) is approx 100].

The **Logistic Regression model** is implemented as a Class which when invoked as an object accepts two parameters, learning rate and the number of epochs. The learning parameter is set to 0.1 as default after performing grid search with various values for improvised performance. The init constructor method initializes two parameters weights, bias along with learning rate and epoch values. The weights parameter is used to hold the weightage which is provided to each feature value during the computations.

Sigmoid Method: The **Sigmoid function** is implemented, which maps input values (the linear combination of weights and features) to a value between 0 and 1, representing the predicted probability of the class.

$$\text{Sigmoid}(x): 1/ (1 + e^{-x})$$

Fit method: The method takes the training data X (features) and Y (labels) and learns the model parameters w (weights) and bias. The method initially sets the weight vector and the bias value to zero. The length of the weight vector is equal to the number of the features. The loop iterates based on the no of epoch provided as input. In each iteration :

1. Computes the Linear combination of features X and the weights with bias.

2. The resultant values is passed through the sigmoid method, it returns the predicted probability.
3. Calculation of the gradients with respect to the loss function. The gradients dw and db is computed.
4. Updates the weights and bias by subtracting it with the product of the learning rate and gradients.

This iterative process continues to adjusts the weights and bias in each epoch to minimize the error between predictions and actual labels.

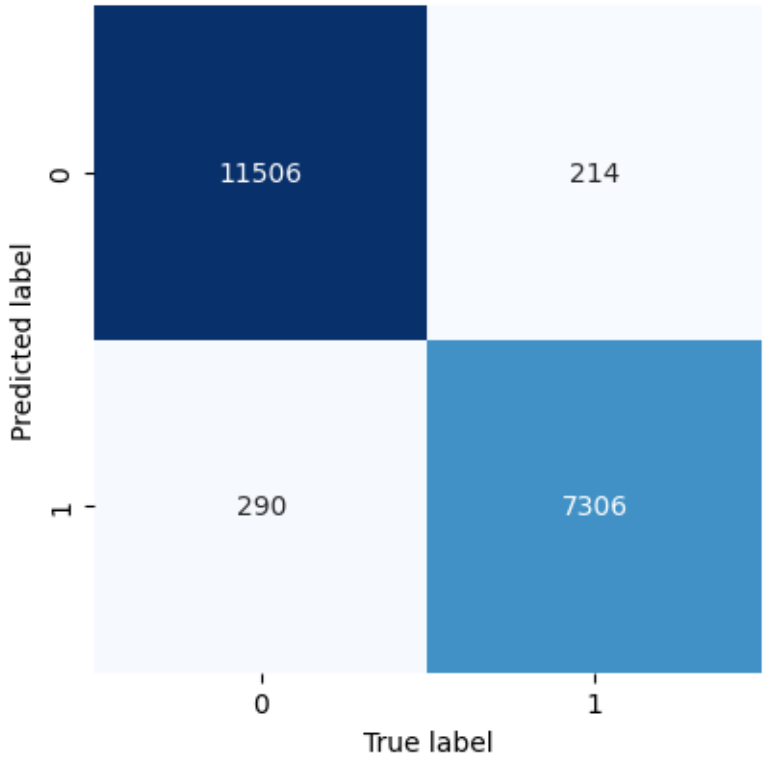
Predict method: The method accepts input features and calculates the predicted probabilities using the trained weights. The linear combination of the features and the trained weights is passed into the sigmoid method to get the predicted probabilities. It returns a vector where i^{th} element is 1 if predicted probability is greater than 0.5 else 0.

getWeights method: The helper method to return the trained weights of logistic regression model.

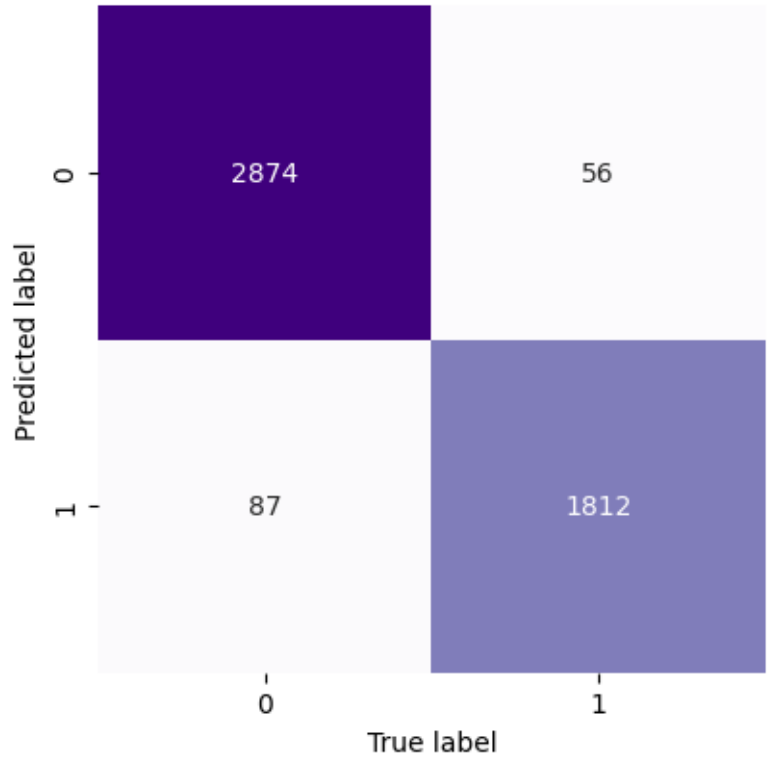
Results:

The model achieves a training accuracy of **97.39%** and a testing accuracy of **97%** on the dataset.

The confusion matrix of the Training set:



The confusion matrix of the Testing set:



The trained model is saved using pickle library along with the trained weight vector and the counter vectorizer.

Two files are submitted ***SpamClassifier.py*** and ***SpamClassifier-Loaded.py*** , the first program is used to build the spam classifier and to train the model on the merged dataset (Enron Spam and Ling-Spam). The second program is used to load the pre trained model and to test the files(text files) from a particular directory.

The SpamClassifier-Loaded.py contains a **class SpamClassifier** which when invoked as an object takes in **the pretrained model** as input argument.

Init method: It loads the pretrained model which is passed as input argument. The method initializes the weight vector with the trained model's weights and initializes the count vectorizer. It loads the stop words of English language which will be used in text preprocessing before the prediction is performed.

Remove_stopwords method: The method accepts text as input argument. It returns the filtered text of the document after removing the stop words present in it.

Preprocess method: The method accepts text as input argument. The text undergoes a sequence of preprocessing steps as mentioned earlier.

The preprocessing steps involve:

1. Converting all the characters to lower case.

2. Removing number, punctuations and special characters from the sentences.
3. The sentences is stripped off the new line characters.
4. The above operations leads to extra spaces between the words in the sentences, these extra spaces are removed.
5. **remove_stopwords** method is invoked to filter the text that do not contribute significant meaning.
6. Lemmatization is applied which reduces words to their base form while preserving their meaning and context.

Sigmoid Method: The **Sigmoid function** is implemented, which maps input values (the linear combination of weights and features) to a value between 0 and 1, representing the predicted probability of the class.

$$\text{Sigmoid}(x): 1/ (1 + e^{-x})$$

Predict method: The method accepts preprocessed text as input argument which is the output of **preprocess method.** It uses count vectorizer's transform method to convert the text into a vector of numerical value, it returns a vector of length equal to the length of the pretrained model's weight vector. The linear combination(dot product) of the feature vector and the weight vector is computed. The **sigmoid method** is invoked to produce the predict probability on the value computed as dot product. The method return 1 if the predicted probability value is greater than 0.5 else 0.

predict method: This is the method which is invoked along with the object of the class created. It accepts the path to the folder **Test** which contains the text files for which the predictions is to be made. It utilizes the glob module to recursively look for .txt files. From each document the text is read, **preprocess method** is invoked on the text which filters the text. The **Predict method** is invoked on this filtered text to get the prediction.

The method returns a vector with predictions for each file i.e, 1 if it is a Spam or 0 if it is not.

A function named Accuracy is implement which takes in the actual labels and the predicted values as input arguments and returns the accuracy value based on the number of correct predictions.

Note: Please invoke the predict method with the class's object and Not Predict method.

Note: The code is written to read the actual labels of text files in the folder Test from a csv file. Please do modify it accordingly for the testing purpose. Please make sure the actual labels are of the form 1 or 0 (type int).

Note: To test the model run SpamClassifier-Loaded.py file. Install nltk and download 'stop words ' before running the file. Install all the necessary libraries imported in the programs.

The model is tested on a kaggle dataset and it report an accuracy of approx. 70%. The model's performance would vary on the test data due to the data shift i.e., the difference between training and operational data distribution. The issue is studied in depth in the following publication:

Jáñez-Martino, F., Alaiz-Rodríguez, R., González-Castro, V. *et al.* A review of spam email detection: analysis of spammer strategies and the dataset shift problem. *Artif Intell Rev* **56**, 1145–1173 (2023). https://doi.org/10.1007/s10462-022-10195-4

Report by,
Manoj Kumar.CM

DA24S018