CS5691 Assignment 3

Pattern Recognition and Machine Learning

Arjun Vikas Ramesh

CS21B008

May 5, 2024

1 The Dataset

- We have chosen the Enron email dataset which is a popular choice of dataset for training spam/ham mail classifiers.
- NOTE: The data contains text of ***latin-1*** (English language) characters alone
- We split the dataset into a 80% train and 20% validation set.

2 Feature Extraction

- Before using the text of data, we need to reduce it to a usable form with prominent features.
- So we do the following steps to extract features from the dataset:
  1. `Convert to lowercase.`
  2. `Remove Punctuations.`
  3. `Extract features like abs@g.com as EMAIL, or www.xyz.com as a LINK, etc to commonly categorize them.`
  4. `Tokenize text, i.e convert it to a list of words.`
  5. `Remove stopwords like "i","me","you", etc`
  6. `Lemmatize the text, i.e. group words that have same core meaning.`
- We explain each of the above substeps below:

2.1 Lower case

  - The purpose of convertion to lowercase is to group all words like ***Go, go, GO,gO*** which all imply the same.

2.2 Remove Punctuations

- We import a list of punctuation using the inbuilt string.punctuation library and remove these from the text.
- This removes the following characters

```
!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~
```

2.3 Extract some common features

- We commonly classify some words which may not have same character, but the same value, using pattern matching.
- NOTE: *We do this prior to removing punctuations since the pattern matching requires punctuation.*
- We pattern match the following:
  1. **MONEY:**
     * We pattern match using regex all such words of form "${NUMBER}", "dollar", "rupee" etc as "MONEY"

```
( \$[\s]*[0-9]+
| dollar
| dollars
| usd
| rupee
| rupees          --> MONEY
| pounds
| pound
| inr
| money )
```

  2. **NUMBER:**
     * We pattern match all numbers to one using regex

```
( [0-9]+ )       --> NUMBER
```

  3. **LINK:**
     * We pattern text of form "{text}.com", "{text}.co.in", "{text}.in", "https://{text}", etc into "LINK"

```
( ([^\s]*(\.com|\.in|\.co\.in|\.co\.uk))
| ((https|http)(\s?):(\s?)/(\s?)/(\s?)[^\s]*) )
  --> LINK
```

4. **EMAIL:**
   * We pattern match all text of form {text}@{text} into EMAIL

```
[^\s]+(\s?)\@(\s?)[^\s]+
  --> EMAIL
```

2.4 Tokenize text

- We extract each word as a separate feature and convert the text into list of words, after doing the above preprocessing.
- We also remove single character words which may arise due to typos.

2.5 Remove Stopwords

- We remove words such as "i", "me", "myself", etc which have no significance in differentiating spam or ham mails.
- We import the stopwords using the ntlk python library, and use the list of stopwords.

2.6 Lemmatization

- Words such as "go", "going", "gone", "goes" all signify the same core meaning "go".
- So we convert all such words to the same using the WordNetLemmatizer from the ntlk library.

- After doing the above pre-processing, we remove Duplicate words from the same email to prevent a few datapoints from altering the classifier. (Will further explain why in upcoming sections)

- We will now create a vocabulary/dictionary which will be used to produce the 0-1 feature vector for each datapoint. (1 if the ith word in dictionary is present in mail else 0)

2.7 VOCABULARY/DICTIONARY

- We create a python dictionary(map) of string to integer which keeps count of number of mails in which the word occurs (The reason to remove duplicate words when tokenizing text).

- We then remove words which occur in less than 1% of emails which will have no significance. This will reduce our number of features greatly. These are words specific to a few emails and is unlikely to help differentiate spam/ham.

- ***NOTE:*** *NUMBER OF EMAILS = 27000*

Minimum number of occurence	Number of features
1	120,000
2	62,500
6	30,000
0.5% of emails	2800
1% of emails	1600

Table 1: feature reduction

2.8 INDEXING

- With the obtained **vocabulary**, we create a mapping of word to index, in order to construct the feature vector.

- With the index mapping, we create the feature vector for each training dataset, 1 if the word in vocabulary is present in email, else 0.

- We require this index mapping in order to convert our test data as well.

3 NAIVE BAYES

- We employ the naive bayes algorithm on the data and obtain the following parameters:

$$p = \frac{\Sigma_{i=1}^{n} y_i}{n} \tag{1}$$

$$p_{spam,j} = \frac{\Sigma_{i=1}^{n} 1(x_j^i = 1 \& y_i = 1)}{\Sigma_{i=1}^{n} 1(y_i = 1)} \tag{2}$$

$$p_{ham,j} = \frac{\Sigma_{i=1}^{n} 1(x_j^i = 1 \& y_i = 0)}{\Sigma_{i=1}^{n} 1(y_i = 0)} \tag{3}$$

- Using the parameters obtained, We calculate the bernoulli probability of the data point being spam or ham, and classify as spam if $P(y = spam|x) > P(y = ham|x)$ or as ham otherwise.

- By using 80:20 train, test split of 34,000 emails, we train the model on 27,000 mails to obtain p, p_{spam}, p_{ham} and the index mapping.

- We test our model by predicting on 7,000 mails (from the test split) and obtain the following accuracy by comparing with Y_{test}.

```
0.9366844602609727
```

4 SVM

- We train our model using the inbuilt SVM module in sklearn.

- Post training, the model obtained gives us an accuracy of:

```
0.9802787663107948
```

- We export the trained model by dumping to a pickle file.

- And for classification, we load the pickle file as a model and classify the mails.

5 FINAL CLASSIFIER:

We use the SVM model since it provides higher accuracy.

6 NOTES:

- The python file included requires a few libraries such as:
  - re (regex for pattern matching)
  - nltk (for lemmatizing the text)
  - string (for punctuations)
  - os (for file opening)
  - numpy (for arrays)
  - scikit-learn (for SVM)

END