Report
A5: Naive Bayes

Sumit Singh
2016PH10575

Best Working Model:

Prediction accuracy on test set: **87.58%**
Training time (dataset of 10000 examples): **5 sec** (Moodle VPL)
Prediction time(test dataset of 40000 examples): **39 sec** (Moodle VPL)

Part (a):

- ➢ Regex was used to split the sentence into unigrams
- ➢ Using **multi-variate Bernoulli event model**, where the entire vocabulary is considered a feature vector, the accuracy was about **83%** on test dataset. Under this model the following formulae were used for calculating the class-conditional probability densities:

$$\phi_{j|y=1} = \frac{\sum_{i=1}^{m} 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\} + 1}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\} + 2}$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^{m} 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\} + 1}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\} + 2}$$

- ➢ However, using **multinomial event model,** where only the size of document is considered a feature vector, the accuracy shot up to above **84%** on test data. Under this model the following formulae were used for calculating the class-conditional probability densities:

$$\phi_{k|y=1} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 1\} + 1}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\} n_i + |V|}$$

$$\phi_{k|y=0} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 0\} + 1}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\} n_i + |V|}.$$

Here,

|V| = the length of vocabulary.
m = size of dataset.
Superscript represents i'th example and subscript denotes the j'th word in it.
n_i = the length (in words) of i'th example.

Part (b):

- ➢ Stemming and stopwords were used to help improve the accuracy.
- ➢ **Stemming** using `nltk.stem.PorterStemmer()` led to a **decrease** in accuracy from 84% to 79%. The reason for this might be the inefficiency of `PorterStemmer()` in stemming words. For example, it considered the word ‘episode’ to be the plural form of ‘episod’ and so, it removed the trailing ‘e’. This approach was clearly wrong, leading to its inefficiency. Thus the use of stemming using `nltk.stem.PorterStemmer()` was avoided.
- ➢ The solution to this might be **Lemmatization** which does morphological analysis of words and it gave better accuracy. However due to the constraints of the assignment, this too was avoided.
- ➢ Using English stopwords, however, led to an **increase** in accuracy to **85%**. Instead of downloading the stopwords everytime the program runs, a better approach was to include the list of stopwords in the program itself. This was done and it led to a decrease in runtime.

Part (c):

- ➢ Bigrams were used as features.
- ➢ In the first approach the all adjacent words pairs were considered bigrams without considering the stopwords. Using this led to an increase in accuracy to **86.98%**.
- ➢ In the second approach the word pairs which contained two stopwords were removed. This led to a further increase in accuracy to **87.5%**.
- ➢ Stemming using `nltk.stem.PorterStemmer()` led to a poor performance in this section as well.