MACHINE LEARNING - 6363

PROJECT 2
NAIVE BAYES CLASSIFIER

Problem

- Classify a document to a respective Newsgroup class
- Extract words from documents and use their appearance frequencies as features for training a Naive Bayes Classifier

Data

- A dataset containing 20,000 Newsgroup messages drawn from 20 Newsgroups is provided
- Link to the dataset: http://www.cs.cmu.edu/afs/cs/project/theo-11/www/naive-bayes.html

- Contains 20,000 documents:
    - Each document contains:
        - Some metadata (Needs to be discarded)
        - Text data useful for identifying the newgroup it comes from
        - The newsgroup (class) label that a document belongs to

Method

The solution to this problem involves three steps: data pre-processing, fitting a Naive Bayes classifier, and testing the model.

1. **Data Pre-processing**

   The data provided is in the form of text documents that contain words that describe the news that was intended to be sent. There are a lot of unwanted texts present that do not provide any useful meaning for our classifier.

   - **Removing unwanted metadata**
     Information at the top of a text document that does not contribute to the

meaning of the news sent and needs to be removed

- **Removing stop words**
  Words that appear frequently but do not have any semantic meaning in a sentence such as [‘i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', … etc]. I tested with
  The stopwords were taken from the ”nltk” corpus. I also tried with other stopwords available online and chose the one that produces the best results.

- **Removing punctuations and special characters**
  The punctuations and special characters do not add any meaning to a text. So, they are removed during preprocessing

2. Naive Bayes Classifier

Naive Bayes classifier has been successful in classifying text documents. It uses probabilistic models to estimate the likelihood that a given document is in a class.

- First, prior probabilities are calculated by accumulating all data from same class and dividing by the number of such instances for each class.
- Calculate word counts from each class and normalize with the respective column counts (i.e. total count for each feature word -- implemented in fit() method)
- Predict the probabilities of each class for the given input vectors. Select the one with the highest property as the classification result.

3. Results

1. Finding the number of feature words to take for the dataset:

n_features	n_reject_words	alpha	test_acc
5000	0	0.68	65.27
6000	100	0.58	67.10
7000	200	0.66	67.98
8000	300	0.66	68.65

9000	400	0.96	68.71
10000	500	0.82	69.51
11000	600	0.87	70.09
12000	700	0.79	70.54
13000	800	0.80	70.74
14000	900	0.87	70.81

- The table above shows the values that were found when I trained the model to find optimal hyperparameters such as **n_features, alpha, and feature words to be selected.**
- From the table, it is clear that as the number of features increases, the accuracy also increases. Also, the most frequent words in the training set are rejected during training.
- So, hyperparameters taken for training the model were:
  N_features = 7000
  Reject_words = 900
  Alpha = 0.85

2. **Finding the optimal value for n_features setting reject_words=900**

n_features	**test_acc**
5000	**68.44**
6000	**69.23**
7000	**70.20**
8000	**69.97**
9000	**70.87**
10000	**70.35**
11000	**70.46**
12000	**70.74**
13000	**71.36**

14000	**71.97**

3. **K-fold Cross Validation (with n_features=14000, reject_words=900)**

Fold --->	Accuracy
0	71.14
1	70.73
2	71.86
3	71.43
4	71.17
5	71.15
6	70.75
7	71.32
8	71.15
9	70.85

Avg Accuracy = 71.15 %

Acc with sklearn's MultinomialNB() classifier = 78.29 %

Compared to the sklearn's classifier, I assume my classifier works well enough.