Spam Filter: Naive Bayes Classifier

Introduction

Email spam filtering is one of the best-know text classification problem. Text classification is the task of classifying documents by their content: that is, by the words of which they are comprised. In email spam filtering: classifying email messages into spam and non-spam (ham).

Document Modeling:

Text classifiers often don't use any kind of deep representation about language: often a document is represented as a bag of words. (A bag is like a set that allows repeating elements.) This is an extremely simple representation: it only knows which words are included in the document (and how many times each word occurs), and throws away the word order!

Consider a document D, whose class is given by C. In the case of email spam filtering there are two classes C = S (spam) and C = H (ham). We classify D as the class which has the highest posterior probability P(C | D), which can be re-expressed using Bayes' Theorem:

$$P(C \mid D) = P(D|C)*P(C)/P(D) \propto P(D|C)*P(C) \qquad (1)$$

In this implementation, we have employed multinomial model, which represents a documents using feature vector whose components corresponds to word types. If we have a vocabulary V, containing |V| word types, then the feature vector dimension is |V|.

Learning Phase:

Given a training set of documents (each labeled with a class) and a set of 2 classes, we can learn a multinomial spam detection model as follows:

1. Define the vocabulary V; the number of words in the vocabulary defines the dimension of the feature vectors.
2. Count the following in the training set:
    - N the total number of documents,
    - N_k the number of documents labeled with class C = {spam, ham},
    - x_{it} the frequency of word w_t in document D_i, computed for every word w_t in V.
3. Estimate the likelihoods of each word given the class- spam/ham.
4. Estimate the class priors i.e. P(Ham)/ P(Spam).

Testing Phase:

1. To classify an unlabeled document D_j, we estimate the posterior probability for each class, P(Ham | D_j) and P(Spam| D_j).
2. Class with higher posterior probability is assigned as label to the unlabeled test document.

Laplace Smoothing:

A drawback of relative frequency estimates— for the multinomial model—is that zero counts result in estimates of zero probability. This is a bad thing because the Naive Bayes equation for the likelihood (7) involves taking a product of probabilities: if any one of the terms of the product is zero, then the

whole product is zero. This means that the probability of the document belonging to the class in question is zero—which means it is impossible.
One way to alleviate the problem is to remove a small amount of probability allocated to observed events and distribute this across the unobserved events. A simple way to do this, sometimes called Laplace's law of succession or add one smoothing, adds a count of one to each word type.

Implementation:

This spam filter has been implemented in 5 files.

Preprocessing Phase

1. Gen_vocabulary.py
   This program generates vocabulary from the training data set.

2. Transform_training_data.py:
   This program transforms the training data containing training label and training instances.

Learning Phase

3. Gen_feature.py:
   This program generate training feature for training instance and also generates parameter of the model which are class prior and conditional probability of words with respect to class spam or ham

Testing Phase:

4. classify_test_data.py:
   This program accepts the complete test data set and predicts the accuracy of the classifier using Laplace smoothing and Naïve Bayes rule.

Driver Program:

5. Naïve_bayes_main.py :
   This program is driver program which can execute all the program previously mentioned in suitable sequence to classify test instances.

Execution:

Implemented classifier can be executed in two ways:

1. Execute the driver program- naïve_bayes_main.py
2. Execute the programs in following sequence
   - Execute Gen_vocabulary.py
   - Execute Transform_training_data.py
   - Execute Gen_feature.py
   - Execute classify_test_data.py

Results:

Implemented naïve Bayes classifier predicts the class of test instance with accuracy 90.2.