ComS 573 Machine Learning
Lab 1
Naive Bayes Classifier for Text Classification

Note:

- Please contact TA if you have difficulty with the lab assignments.
- The lab description is based on Java. However feel free to use other programming languages such as Python. Be sure to provide a good README file explaining how to run your program.
- In this lab assignment, you are required to write your own code to implement the Naive Bayes algorithm. You are not allowed to use the existing implementation of the Naive Bayes algorithm in machine learning packages. However, if you wish, you may use machine learning packages such as WEKA to check the correctness of your results.

Naive Bayes classifiers have been successfully applied to classifying text documents. In this lab assignment, you will implement the Naive Bayes algorithm to tackle the "20 Newsgroups" classification problem.

1 Data Set

The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. It was originally collected by Ken Lang, probably for his `Newsweeder: Learning to filter netnews` [1] paper, though he did not explicitly mention this collection. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.
The data is organized into 20 different newsgroups, each corresponding to a different topic. Here is a list of the 20 newsgroups:

`alt.atheism`
`comp.graphics`
`comp.os.ms-windows.misc`
`comp.sys.ibm.pc.hardware`
`comp.sys.mac.hardware`
`comp.windows.x`
`misc.forsale`
`rec.autos` `rec.motorcycles`
`rec.sport.baseball` `rec.sport.hockey`
`sci.crypt`
`sci.electronics`
`sci.med`
`sci.space`
`soc.religion.christian`
`talk.politics.guns`
`talk.politics.mideast`
`talk.politics.misc`
`talk.religion.misc`

The original data set is available at `http://qwone.com/~jason/20Newsgroups/`. In this lab, you won't need to process the original data set. Instead, a processed version of the data set is provided (see `20newsgroups.zip`). This processed version represents 18824 documents which have been divided into two subsets: training (11269 documents) and testing (7505 documents). After unzipping the file, you will find six files: `map.csv, train_label.csv, train_data.csv, test_label.csv, test_data.csv, vocabulary.txt`. The `vocabulary.txt` contains all distinct words and other tokens in the 18824 documents. `train_data.csv` and `test_data.csv` are formatted `"docIdx, wordIdx, count"`, where `docIdx` is the document id, `wordIdx` represents the word id (in correspondence to `vocabulary.txt`) and `count` is the frequency of the word in the document. `train_label.csv` and `test_label.csv` are simply a list of label id's indicating which newsgroup each document belongs to (with the row number representing the document id). The `map.csv` maps from label id's to label names.

2 What You Will Do

In general, you will implement a Java program that takes the input files, builds a Naive Bayes classifier, and outputs relevant statistics. Call your code as follows:

```
java NaiveBayes training_label.csv training_data.csv testing_label.csv
testing_data.csv
```

(Note: you may use a different name for the Java class.)
You will learn your Naive Bayes classifier from the training data (`train_label.csv`, `train_data.csv`), then evaluate its performance on the testing data (`test_label.csv`, `test_data.csv`). Specifically, your program will accomplish the following two tasks.

2.1 Learn the Naive Bayes Model

You will implement the *multinomial model* ("*bag of words*" model) discussed in the lecture. In the learning phase, you will estimate the required probability terms using the training data.

For each target value ω_j (each newsgroup)

- Calculate class prior $P(\omega_j)$
- Calculate n: total number of words in all documents in class ω_j (i.e., total length)
- For each word w_k in Vocabulary
  - Calculate n_k: number of times word w_k occurs in all documents in class ω_j.
  - Calcualte Maximum Likelihood estimator $P_{MLE}(w_k|\omega_j) = \frac{n_k}{n}$ and Bayesian estimator $P_{BE}(w_k|\omega_j) = \frac{n_k+1}{n+|Vocabulary|}$ (this is Laplace estimate).

Your program should output the class priors, for example (the following numbers are arbitrary, not true values)

```
P(Omega = 1) = 0.0426
P(Omega = 2) = 0.0516
P(Omega = 3) = 0.0508
        ...
```

You need not print out all $P_{MLE}(w_k|\omega_j)$ or $P_{BE}(w_k|\omega_j)$, just take a look at the values of these two types of estimators. *What do you observe? Discuss what you have observed in your report.*

2.2 Evaluate the Performance of the Classifier

In this task, you will evaluate your Naive Bayes classifiers that you have learned in Section 2.1 on both the training and the testing data. You will use your Naive Bayes classifiers to make classification decision on these data set and calculate relevant statistics such as overall accuracy, class accuracy, confusion matrix, etc. When making classification decision, consider only words found in Vocabulary. Let *positions* be all word positions in a document $(x_1, x_2, \ldots)$ that contain tokens found in Vocabulary, return ω_{NB} where

$$\begin{aligned} \omega_{NB} &= argmax_{\omega_j} P(\omega_j) \prod_{i\ in\ positions} P(x_i|\omega_j) \quad (1) \\ &= argmax_{\omega_j} P(\omega_j) \prod_{k=1}^{|Vocabulary|} P(w_k|\omega_j)^{N_k}. \quad (2) \end{aligned}$$

You may find it more convenient to compute using logarithms (to prevent underflow), e.g.:

$$\omega_{NB} = argmax_{\omega_j} [\ln P(\omega_j) + \sum_{i\ in\ positions} \ln P(x_i|\omega_j)]. \quad (3)$$

Note however that $P(x_i|\omega_j)$ or $P(w_k|\omega_j)$ terms may be zero for MLE.

2.2.1 Performance on Training Data

First you will evaluate your Naive Bayes classifier on the training data, for both Bayesian and ML estimators respectively. After making prediction for each document in the training data set, you will calculate and output the *overall accuracy*, i.e, the percentage of correctly classified documents

$$Overall\ Accuracy = \frac{number\ of\ correctly\ classified\ documents}{total\ number\ of\ documents}, \quad (4)$$

as well as the *class accuracy = (predicted as class i)/(number of documents in class i)* for each newsgroup. A sample output is given as follows (the numbers are arbitrary, not true values).

```
Overall Accuracy = 0.9211
     Class Accuracy:
    Group 1:  0.9667
    Group 2:  0.9191
          ...
```

Further, you will construct and output the confusion matrix (`https://en.wikipedia.org/wiki/Confusion_matrix`). An example is as follows (the numbers are arbitrary, not true values):

```
464    0    0    0     ...
  1  534    6   15     ...
  1   10  503   23     ...
  0   10    4  546     ...
           ...
```

A cell (i, j) in the matrix represents the number of documents in group i that are predicted to be in group j.

2.2.2 Performance on Testing Data

Now you will evaluate your classifiers on the testing data set. Repeat the experiments described in Section 2.2.1 but on the testing data set. Compare the results obtained with the results you have obtained in Section 2.2.1. What do you observe? Discuss.

Also compare the results obtained using the Bayesian estimators against the Maximum Likelihood estimators. What do you observe? Which one is better? Discuss.

2.3 Summary

In summary, your program will output the following relevant statistics:

- Class priors
- Performance on training data (for both Bayesian estimators and ML estimators): overall accuracy, class accuracy, confusion matrix.
- Performance on testing data (for both BE and MLE): overall accuracy, class accuracy, confusion matrix.

3 What to Turn In

Turn in via Canvas a compressed file (.zip) containing the following:

- All of your commented source code (`.java` files). (For Java code, make sure the debugging flag is set to false. **Do not turn in the results of running the code with `debugging = true`**).
- A `README` file explaining how to compile and run the program.
- A short lab report that includes the experimental results obtained and answers to questions.

References

[1] Ken Lang, Newsweeder: Learning to filter netnews, Proceedings of the Twelfth International Conference on Machine Learning, 331-339 (1995).