hw1: Naïve Bayesian Learning

- **Kai Huang**
- k.huang@pitt.edu

1. Introduction

I implemented Naïve Bayes classifier using Python to classify the 20 newsgroups dataset, which contains 20 topics, 1000 texts per topic. The whole process includes getting lexicon (namely, dictionary), data pre-processing, training and testing. And I also did a brief review for my result.

1.1 To run the code

I use Python 3.x. The code contains the entire process, so it may cost time to get the final result.

Extract data from zip --> Split data into training set and testing set --> Training --> Testing

Command Line

1. `pip3 install zipfile` (if you don't install this module)
2. cd to the code folder path
3. `python main.py`

2. Implementation

The followings introduce technical details of implementation for the classification task. I use **Python 3.x** as my coding language without any non-built-in modules or packages to implement this algorithm.

2.1 Preparing Lexicon

I downloaded the lexicon from the link https://github.com/first20hours/google-10000-english/blob/master/google-10000-english-usa-no-swears-long.txt. It contains a list of about 10,000 most common English words in order of frequency, which is collected by Google.

The original lexicon is comprehensive and contains many stopwords, e.g., "a, an, the, etc", which are meaningless in semantics. I removed these words from the lexicon by Python.

2.2 Data Pre-processing

As required in the instruction, the dataset is divided into half training set and half testing set. I use Python randomly spliting the data into `\Train` and `\Test` folders.

To easily handle the data, I create two files `train.csv` and `test.csv` which record the mappings of every sample's path to its class label.

2.3 Training

I use bag of words model. According to Naïve Bayes rule for classification,

$$h_{NB}(x) = \arg\max_{y} P(y) \prod_{i=1}^{LengthDoc} P(x_i|y) \tag{5}$$

where $P(y)$ is the frequency of topic y texts in the training set, which is defined by

$$P(y) = \frac{\#\text{ texts in topic } y}{\#\text{ total texts}} \tag{1}$$

$P(x_i|y)$ means, the frequency of word x_i in all texts of topic y, which is defined by

$$P(x_i|y) = \frac{\#\text{ word } x_i \text{in topic } y}{\#\text{ total words in topic } y} \tag{2}$$

For robustness, I use $\log(\cdot)$ to modify (1)

$$h_{NB}(x) = \arg\max_{y}\{\ \log P(y) + \sum_{i=1}^{LengthDoc} \log P(x_i|y)\ \} \tag{3}$$

And for (3)

$$P(x_i|y) = \frac{\#\text{ word } x_i \text{in topic } y\ +\alpha}{\#\text{ total words in topic } y\ +1}\ ,\ \ \alpha \text{ is very small} \tag{4}$$

The aim for training is to get $P(y)$ and $P(x_i|y)$ for each y and each x_i. My code scans all the training texts to count these numbers, and record these result in `Pxy.csv` and `Py.csv`.

`Pxy.csv` records a ($\#\text{ topic} \times \text{LengthLexicon}$) matrix.

`Py.csv` records a ($\#\text{ topic} \times 1$) matrix.

2.4 Testing

For a certain testing text X, we scan every word in X. For each possible topic y, we retrieve y from `Py.csv`. For each word x_i in X, we retrieve $P(x_i|y)$ from `Pxy.csv`. Then follow rule (1), we can do prediction.

2.5 Tricks

There are some characters in the text data can't be decoded by Python character encoding standard, which can raise error. So I just intentionally ignore these characters since they are useless.

Before doing counting in **2.4 Training**, we need to split all the texts into one-by-one words. In order to successfully do so, I first scan all the words, replace non-alphabetical words with space, convert uppercase letters into lowercase. Then I call `.split()` in Python to get a list of split words.

We can do stemming for every word in lexicon and texts as pre-processing. e.g., we can unify words `swimming, swam, swimmer` by `swim` in order to signify the feature. To do stemming, we need use `nltk package`, which is not a Python built-in package.

3. Result

3.1 Testing Result

I tested the model in 3 different settings:

- **Setting 1:** Remove stopwords and any words with its length less than 3 in the lexicon.
- **Setting 2:** Setting 1 + stemming.
- **Setting 3:** Baseline. No lexicon refinement, no stemming.

Testing Accuracy

Topic	Setting 1	Setting 2	Setting 3
alt.atheism	0.70	0.70	0.71
comp.graphics	0.71	0.71	0.74
comp.os.ms-windows.misc	0.71	0.71	0.43
comp.sys.ibm.pc.hardware	0.73	0.70	0.76
comp.sys.mac.hardware	0.78	0.73	0.80
comp.windows.x	0.82	0.79	0.86
misc.forsale	0.75	0.76	0.79
rec.autos	0.89	0.87	0.88
rec.motocycles	0.92	0.90	0.93
rec.sport.baseball	0.91	0.87	0.91
rec.sport.hockey	0.96	0.95	0.96
sci.crypt	0.92	0.90	0.92
sci.electronics	0.82	0.80	0.83
sci.med	0.90	0.86	0.91
sci.space	0.93	0.93	0.93
soc.religion.christian	0.96	0.95	0.96
talk.politics.guns	0.88	0.88	0.88
talk.politics.mideast	0.89	0.85	0.87
talk.politics.misc	0.74	0.70	0.75
talk.religion.misc	0.59	0.61	0.59
Test Acc Avg	**0.83**	**0.81**	**0.82**

3.2 Result Review

By the way, the training accuracies of three setting are all about 0.95.

The testing accuracy will get slightly better with lexicon refinement, however, stemming will lower the testing accuracy.

By looking at the accuracy of every topic, we can find topic `talk.religion.misc` is always the lowest. If we get a closer look at the texts in this topic, we will find `talk.religion.misc` newsgroup does not typically focus on some relatively specific topics, they may talk about many more other field of interest.