CSC 425/525 Artificial Intelligence
Project 2: Spam Email Filter

Semester: Fall 2020

Instructor: Dr. Junxiu Zhou

Introduction

We've learned different ways of knowledge representation and problem-solving strategies to answer different queries. In this project, we tried to answer query using a classification strategy. That is given a set of classes, we seek to determine which class(es) a given object belongs to. To conduct the classification task, we need to find a rule that captures a certain combination of keywords that indicates a unique class. To achieve this, we can use hand-coded rules which have good scaling properties, but creating and maintaining them over time is labor intensive. Or we can build an expert system to create rule sets that will rival or exceed the accuracy of the automatically generated classifiers as we plan to do in this project; however, it can be hard to find the expert with this specialized skill.

Alternatively, we will practice to use statistical instead of knowledge representation strategy to represent text information. A statistical text classification, we require a number of good example documents (training documents) for each class. The need for manual classification is not eliminated because the training documents come from a person who has labeled them, for example to annotate an email as spam or ham. But labeling is arguably an easier task than writing rules.

A typical spam filter is a program that is used to detect unsolicited and unwanted email and move those emails to a user's spam inbox folder. Like other types of filtering programs, a spam filter looks for certain criteria on which it bases judgments. Such as, the simplest and earliest can be set to watch for particular words in the subject line of messages and to exclude these from the user's inbox, as shown in the Fig. 1.

Fig. 1 illustration of spam email filter.

Methodology

1. Get to know the dataset

In this project, the dataset has been divided into training set, i.e., "train-mails" which contains 702 email text files, and test set, i.e., "test-mails" which contains 260 email text files. Note that, in the dataset, the spam email samples are named with "spmXXX". The detail information of this dataset is listed in Table 1.

Table 1 Spam Filter Project Dataset

Dataset	train-mails (702)		test-mails (260)	
	ham emails	spam emails	ham emails	spam emails
	351	351	130	130
label	0	1	0	1

The name of this dataset is Ling-Spam Corpus which can be download here https://www.stat.purdue.edu/~mdw/598/datasets.html. Note that, the provided dataset in this project is a preselected subset of the original dataset.

In this project, we use $d \in D$ represents one email text file, where D is all the email files. $C = \{c_1, c_2\} = \{0, 1\}$ represent ham, spam classes, respectively. Then for the email files in the training set we have the following file and label pair,

$$< d, c >$$

For example,

$< d, c >=<$ listserv $international\ conference\ \ldots\ please\ contact\ organizer, ham >$

To implement this in Python, we have to read all email files in, and then extract content in each file and store it. For this purpose, we can define a method named *read_file*.

2. Feature Extraction

Text is very common information in our daily life, however it is hard to direct use text conduct computing in programs. You can think of text as a sequence of characters, words, phrases and named entities, sentences, paragraphs, and so on. It seems natural to think of a text as a sequence of words. A word is a meaningful sequence of characters, and in English we can split a sentence by spaces or punctuation, for example,

Input Text: "anyone point book book article causative construction Korean book"

Output Words: anyone point book article causative construction korean

One thing you may noticed that the input text in our dataset is a little different from the email that we see. This because these files are preprocessed by using basic natural language

preprocessing technologies, such as Tokenization, Stemming, and Lemmatization. If you are interested, you are welcome to explore more.

In this project, we use the bag of words (BOW) feature for the text. BOW try to count occurrences of a particular word/token in our text, then the higher the occurrence the more important word in the text. For each token or word, we will have a feature column, this is called **text vectorization**. The problem of the BOW is that we loose word order, hence the name "bag of words".

For example, the input text above can be represented by a BOW vector

anyone	point	book	article	causative	construction	korean
1	1	3	1	1	1	1

Here the output words together work as a dictionary for us to create the vector. For the spam email filter, ham emails and spam emails may use different words to covey information. For example, for spam emails there may be words heavily occurrence, such as, discount, money, shopping etc., while for ham emails there may be words heavily occurrence, such as, work, time, meetings, and so on. So, if we can take all unique words from the training set to form a dictionary, then we can use text vectorization to represent each of the email file. However, the dictionary may contain too many unique words, and some of these words may not convey useful meanings, such as anyone, point, is, etc. In this case, we sort the words in the dictionary based on their frequency, and choose n of them to form a new dictionary which contains a subset of the original dictionary. In this project we choose $n = 3000$ most important words in the dictionary. Then we can use one text vector to represent each email file.

3. Classification Model

With the provided training data and related label, we then wish to learn a classifier or classification function f that maps email file to corresponding class label:

$$f: D \rightarrow C$$

This type of learning is called supervised learning, because a supervisor (the human who defines the classes and labels training documents) serves as a teacher directing the learning process. Once we have learned the classification function f, we can apply it to the test set (or test data). Our goal in this project is to have a classifier to get high accuracy on test data or new data. Accuracy means the classifier successfully classifies the test sample number over all the test set samples. Normally, it is to achieve high accuracy on the training set (e.g., we can simply memorize the labels). But high accuracy on the training set in general does not mean that the classifier will work well on new or unseen data in an application. However, when we use the training set to learn a classifier for test data, we make the assumption that training data and test data are similar or from the same distribution.

In this project, we mainly focused on two types of classification model, i.e., Multinomial Naïve Bayes and Bernoulli Naïve Bayes. In addition, if you're interested in Gaussian Naïve Bayes statistic model, you can opt to finish the bonus project task.

3.1 Multinomial Naïve Bayes (NB) Statistic Model

The first supervised learning method we used is the **multinomial Naive Bayes** or **multinomial NB** model, a probabilistic learning method. It is relatively suitable for text related classification. The original probability equation is:

$$P(h_i|E) = \frac{P(E|h_i)*P(h_i)}{P(E)} \quad (1)$$

E is the given evidences set (which is the features in a specific classification problem); h_i is one hypothesis in H (H is the set of all hypotheses, which is all classes in a specific classification problem). $P(E|h_i)$ is the conditional probability of term E belongs to class h_i. $P(h_i)$ is the prior probability of an email text belonging to class h_i.

We assume that P(E) is the same for all the emails, we can approximate the equation using:

$$P(h_i|E) = P(E|h_i) * P(h_i) \quad (2)$$

Since the given evidence E contains a sequence of n_E words $e_1, e_2, \dots, e_{n_E}$, these words are independent from each other in BOW, therefore Eq. (2) can be rewrite as

$$P(h_i|E) = \prod_{1\le k\le n_E} P(e_i|h_i) * P(h_i) \quad (3)$$

where e_k is one evidence (i.e., a word term) in E that are part of the vocabulary we use for classification. n_E is the number of evidences in E (i.e., total words term in an email file). For example, the $< e_1, e_2, \dots, e_{n_E} >$ for the input text in Section 2 might be $< anyone, point, book, article, causative, construction, korean >$ with $n_E = 7$.

$P(h_i|E)$ means given a feature vector E of an email text file, we want to calculate the probability that this feature vector belongs to ham $P(h_0 = ham|E)$ and the probability that this feature vector belongs to spam $P(h_1 = spam|E)$. In this project, our goal is to find the best class for the email file. The best class in NB classification is the most likely or **maximum a posteriori** (MAP) class h_{best}:

$$h_{best} = \underset{h_i\in H}{\operatorname{argmax}} \prod_{1\le k\le n_E} P(e_k|h_i) * P(h_i) \quad (4)$$

There is one problem in this approximation Eq. (4), that is the multiplicative of probabilities can cause the float point overflow problem. So, we perform the computation by adding logarithms of probabilities instead of multiplying probabilities to calculate the parameterized probabilistic distribution:

$$h_{best} = \underset{h_i\in H}{\operatorname{argmax}} \sum_{1\le k\le n_E} \log P(e_k|h_i) + \log P(h_i) \quad (5)$$

Note that the class with the highest log probability score is still the most probable, and the logarithm function is monotonic. The interpretation of Eq. (5) are: the conditional probability $P(e_k|h_i)$ indicates how good a word is for class h_i; the priori probability $P(h_i)$ is a weight that indicates the relative frequency of class h_i; the sum of the log prior and term weights is then a measure of how much evidence there is for the email file being in the class, and the Eq. (5) will select the class for which we have the most evidence.

The question is how do we estimate the terms $P(e_k|h_i)$ and $P(h_i)$? We can estimate them from the training set:

For the prior estimate is:

$$P(h_i) = N_{h_i}/N \tag{6}$$

where N_{h_i} is the number of email files in class h_i in the training set and N is the total number of email files in the training set.

For the conditional probability $P(e_k|h_i)$, we estimate it as the relative frequency of term e_k in files or documents belonging to class h_i,

$$P(e_k|h_i) = \frac{O_{h_i,e_k}}{\sum_{e_t \in Dictionary} O_{h_i,e_t}} = \frac{O_{h_i,e_k} + \alpha}{\sum_{e_t \in Dictionary} O_{h_i,e_t} + n * \alpha} \tag{7}$$

where O_{e_k} is the number of occurrences of e_k in the training email files from class h_i. $\sum_{e_t \in Dictionary} O_{h_i,e_t}$ represents the sum of occurrences of every word in the dictionary from class h_i in the training email files. As defined in Section 2, n is the number of words in the dictionary. Note that, here α is a smooth variable used to eliminate zeros, and we normally set $\alpha = 1$, called add-one smoothing.

To implement multinomial NB model in Python:

1. Preparing the training data
    a. Store the email files based on their class labels (either ham or spam)
    b. For each class, record the number of files and the distinct or unique words in each document
    c. Generate the feature matrix for each file, the occurrence of the word in the dictionary is recorded, then the feature matrix has the following format

feature matrix=$\begin{bmatrix} & word1 & word2 & \dots \\ file1 & O_{11} & O_{12} & \dots \\ file2 & O_{21} & O_{22} & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix}$

2. Generate the parameterized distribution based on the feature matrix for each class to form the logarithm matrix

feature_log_prob=$\begin{bmatrix} & word1 & word2 & \dots \\ spam & \log p(e_1|spam) & \log p(e_2|spam) & \dots \\ ham & \log p(e_1|ham) & \log p(e_2|ham) & \dots \end{bmatrix}$

then the multinomial NB model parameters can be achieved.

To test the multinomial NB model in Python:

To verify the effectiveness of the build multinomial NB model, we need to perform the following steps:

1. For each email file in the test set, we calculate its related feature vector based on the word in the dictionary.
   feature vector of the test email file $= \begin{bmatrix} & word1 & word2 & \cdots \\ testfile & O_{t1} & O_{t2} & \cdots \end{bmatrix}$

2. Do the elementwise product with *feature_log_prob* for both spam and ham class followed by the equation in (5).
   For ham class probability:
   $$h_{0\_test} = \sum_{1 \le k \le n_E} \log P(e_k|h_0) * O_{te_k} + \log P(h_0)$$
   For spam class probability:
   $$h_{1\_test} = \sum_{1 \le k \le n_E} \log P(e_k|h_1) * O_{te_k} + \log P(h_1)$$
   If $h_{0\_test} > h_{1\_test}$, then the test email file belongs to the ham class;
   Otherwise, the test email file belongs to the spam class.

3.2 Bernoulli Naïve Bayes (NB) Statistic Model (optional)

The second way to set up an NB classifier is to use the Bernoulli NB model. The Bernoulli model has the same time complexity as the multinomial model. However, the Bernoulli NB model is a binary independence model, which generates an indicator for each term of the vocabulary, either 1 indicating presence of the term in the document or 0 indicating absence. The different generation models imply different estimation strategies and different classification rules. More information about the Bernoulli model can be find here (https://en.wikipedia.org/wiki/Bernoulli_distribution).

In general, in this project most of the parts of calculation of Bernoulli NB model is similar to Multinomial NB model. But still there are some differences. The differences are:

a) We need to transfer each term of the feature matrix to 1 and 0 since it follows the Bernoulli distribution. If the original entry is non-0 then it is 1 or it is 1.

b) feature matrix$= \begin{bmatrix} & word1 & word2 & \ldots \\ file1 & 1 & 0 & \ldots \\ file2 & 0 & 1 & \ldots \\ \ldots & \ldots & \ldots & \ldots \end{bmatrix}$

c) When conducting the prediction, the multiplication will be changed to:

spam = $\sum_{1\le k\le n_E} \log p(e_k|spam) * (0\ or\ 1) + (1 - \log p(e_k|spam)) * (flip\ of\ 0\ or\ 1) + \log p(h_1)$

ham = $\sum_{1\le k\le n_E} \log p(e_k|ham) * (0\ or\ 1) + (1 - \log p(e_k|ham)) * (flip\ of\ 0\ or\ 1) + \log p(h_0)$

3.3 Gaussian Naïve Bayes Statistic Model (Optional)

Though Gaussian Naive Bayes is suitable for continuous data, we want to implement here to see the effect of this model. A general Gaussian distribution is used:

$$f(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{8}$$

What we will do here is:

a) Preparing the training data and do statistical according to class and features. This process is similar to previous two algorithms.

b) For a Gaussian model, we want to get the mean and standard deviation:

$$\text{Mean } \mu = \frac{\sum_{j\in N_{e_k}} N_j}{N_{e_k}};\ \text{Std } \sigma = \sqrt{\frac{\sum_{j\in N_{e_k}} (N_j-\mu)^2}{N_{e_k}}} \tag{9}$$

This model is for each feature (in our instance for each word) in one class.

c) Prediction is simple sum all features in testing sample according to equation (8).

For each entry in testing feature matrix set $x = entry$ and calculate the result.

Project Requirement

In this project, we need to use different statistical models that can be used to perform spam email classification task. The detailed requirements are:

1. Using multinomial NB model to solve the spam email filter problem (80%)

As stated in section 3.1.

2. Result analysis. (10%)

Please test your probability model performance. Try to understand the computation cost, accuracy. Write down your detailed analysis and conclusion.

3. Project Report (10%)

You have to write a project report for this project. You have to complete the template and submit it together with your project source code. The length of the report must longer than 1 page. The template of the project report is attached in the project zip file.

4. Bonus Works (Extra 30%)

A. Using Bernoulli Naïve Bayes model to solve the spam email filter problem (10%)

   As stated in section 3.2.

B. Using Gaussian Naïve Bayes Statistic Model to solve the spam email filter problem. (10%)

   As stated in section 3.3.

C. Increasing the training and testing set size and see the results. Is increasing the size of samples have a positive influence on the accuracy. (10%)

Hand-In Requirements

(1) The starter code is provided in Python language. You must follow the provided starter code to finish this project. There are two reasons for this:
   a. To prevent plagiarism;
   b. Reading and understanding other people's code should be one the ability of CS students.

   Note that if your team choose to use Java to implement this project, please contact me for the information about the starter code.

(2) If you completed this project, you need to hand in the **source code** been completed by you or your group. Your source code compressed to a single ZIP file. The name of the code archive should be **SEFilter_YourTeamname.zip.**

The code should be well commented, and it should be easy to see the correspondence between what's in the code and what's in the report. You don't need to include executable or various supporting files (e.g., utility libraries) whose content is irrelevant to the assignment. If the instructor finds it necessary to run your code in order to evaluate your solution, the instructor will get in touch with you.

(3) A project report in PDF format

- The report should briefly describe your implemented solution and fully answer all the questions posed above. **Remember**: *you will not get credit for any solutions you have obtained, but not included in the report*.
- All group reports need to include paragraph of individual contribution, i.e., which group member was responsible for which parts of the solution and submitted material. The instructor reserves the right to contact group members individually to verify this information.
- Describe the solution and compare the number of attempted search steps and execution time for your uninformed search method and informed search method implementations.
- The name of the report file should be **SEFilter_YourTeamname.pdf**. Don't forget to include the names of all group members and the number of credit units at the top of the report. (*You can find the report template on the provided zip file together with the starter code.*)
- Finally, which part you think can be improved. Or the challenges you encountered during this project.

Note that, the instructor reserves the right to give **bonus points** for any advanced exploration or especially challenging or creative solutions that you implement. **If you submit any work for bonus points, be sure it is clearly indicated in your report.**

WARNING: You will not get credit for any solutions that you have obtained, but not included in your report!