谭树杰 11849060

CS405 Machine Learning

Lab # Bayes Report

Lab (100 points):

Text mining (deriving information from text) is a wide field which has gained popularity with the huge text data being generated. Automation of a number of applications like sentiment analysis, document classification, topic classification, text summarization, machine translation, etc has been done using machine learning models. In this lab, you are required to write your spam filter by using naïve Bayes method. This time you should not use 3rd party libraries including scikit-learn.

Methodology:

The number of spam mails and ham mails in the training dataset are equal, so according to the Bayes's theorem, we have:

$$\Pr(spam|word) = \frac{\Pr(word|spam)}{\Pr(word|spam)+\Pr(word|ham)}$$

where:

- Pr(spam | word) is the probability that a message is a spam, knowing that the word "word" is in it;
- Pr(word | spam) is the probability that the word "word" appears in spam messages;
- Pr(word | ham) is the probability that the word "word" appears in ham messages.

Obviously, we should not determine a mail is spam or ham by simply considering just one word. Therefore, we combine several individual probabilities to enhance the performance. If the words in the mail are independent events, we can derive the following formula:

$$p = \frac{p_1 p_2 \dots p_N}{p_1 p_2 \dots p_N + (1-p_1)(1-p_2)\dots(1-p_N)}$$

Where:

- p is the probability that the given mail is a spam, give N words;
- p_i is the probability that it is a spam given $word_i$ ($i = 1,2,\dots N$)

We can observe that $p_1 p_2 \dots p_N$ is suffer to floating-point underflow. Hence, we use the following formula to computer the probability:

$$p = \frac{1}{1+e^{\eta}}$$

Where:

$$\eta = \sum_{i=1}^{N} [\log_e(1 - p_i) - \log_e p_i]$$

The implementation of Bayes classifier in python is shown at the end of the report, and whole code is in Bayesian.py

Results:

If p is larger than a predefined threshold, than we regard the mail as a spam mail. The performance of difference choose of threshold is shown in the table below.

Threshold	0.99	0.95	0.90	0.80
accuracy score	0.984615	0.984615	0.988461	0.988461
recall score	0.976923	0.976923	0.984615	0.984615
F-1 score	0.984496	0.984496	0.988416	0.988416

Analysis:

As we can see from the table above, the algorithm is pretty simple but performance is pretty good. However, the conditional independence of words is rare true in real world.
As the threshold increases, the recall score decreases, due to that some mails may seems like spam mail. As threshold increases from 0.90 to 0.95, the accuracy score and F-1 score decreases, which implies the threshold should not be too large.

Conclusion:

In this report, I implement the naïve bayesian classifier roughly. The algorithm is simple but surprising performance is achievement. We could adjust the threshold for different requirements of accuracy score and recall score. Obviously, there's lots of room for improvement, such as the frequency of word in mail should be considered.

```
class Bayes:
    def __init__(self):
        self.length = -1        # length of feature
        self.vectorcount = dict()   # vectorcount[1] indicates feature vector of spam mails
    def fit(self, feature_matrix: list, labels: list):
        if len(feature_matrix) != len(labels):
            raise ValueError("the length of feature_matrix is not equal to the length of labels")
        self.length = len(feature_matrix[0])
        self.vectorcount[0] = []
        self.vectorcount[1] = []
        for vector, label in zip(feature_matrix, labels):
            self.vectorcount[label].append(vector)
        print("training finished")
        return self
    def predict_one(self, test_feature):
        if self.length == -1:
            raise ValueError("NO training")
        # calculate the probability of test_feature belong to spam or ham
        ham_vector = np.array(self.vectorcount[0]).T
        spam_vector = np.array(self.vectorcount[1]).T
        eta = 0
        for index in range(0, len(test_feature)):
            if test_feature[index] == 0:
                continue
            word_ham = list(ham_vector[index]).count(test_feature[index])
            word_spam = list(spam_vector[index]).count(test_feature[index])
            p = word_spam / float(word_spam + word_ham)
            if p == 1:
                p -= 0.00001
            else:
                p += 0.00001
            eta += np.log(1-p) - np.log(p)
        prob = expit(-eta)
        label = 0
        if prob > 0.95:
            label = 1
        return label
    def predict(self, test_feature_matrix):
        predict_labels = []
        for feature in test_feature_matrix:
            # print(feature)
            predict_labels.append(self.predict_one(feature))
        return predict_labels
```