深 圳 大 学 实 验 报 告

课程名称：概率论与数理统计

实验项目名称：**Naive Bayes Spam Filtering Experiment**

学院：电子与信息工程学院

专业：电子信息工程

指导教师：陈昌盛老师

报告人：贾苏健 学号：**2022280485**

班级：**06**

实验时间：**2023.11.16**

实验报告提交时间：**2023.11.23**

教务处制

Aim of Experiment:

1. This experiment is based on the Naive Bayes algorithm to implement spam email classification. Bayes' theorem is used to calculate the conditional probability of an event occurring given prior information. In spam email classification, we consider text data as a collection of individual words. Using the Naive Bayes algorithm, we can calculate the probability that an email is spam.

$$p(\text{Spam}|w_1,\ldots,w_n) = \frac{p(w_1,\ldots,w_n|\text{Spam})p(\text{Spam})}{p(w_1,\ldots,w_n)}$$

2. The key assumption of Naive Bayes is that each word is independent when classified, known as the Naive Bayes assumption. This simplifies the calculation of conditional probability but is an oversimplification in reality.

$$p(\text{Spam}|w_1,\ldots,w_n) = \frac{p(\text{Spam})\prod_{i=1}^{n} p(w_i|\text{Spam})}{p(w_1,\ldots,w_n)}$$

3. In the Naive Bayes algorithm, to improve numerical stability, we use the method of logarithmic probabilities. By performing operations on the logarithm of probabilities, operations involving the multiplication of multiple probabilities are transformed into addition, avoiding the issue of rounding to zero in computer

calculations. This processing method simplifies calculations while still effectively making predictions without the need for reverse operations.

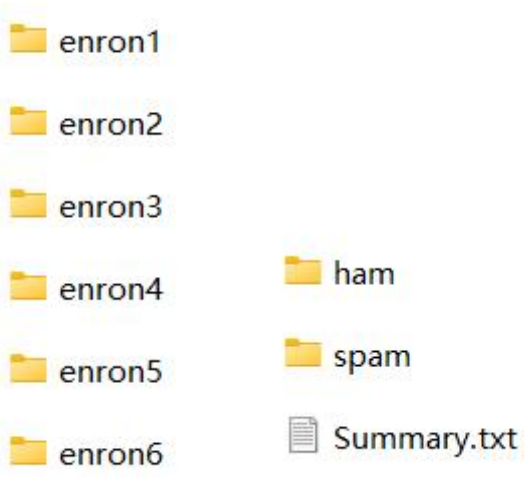
$$\log p(\text{Spam}|w_1,\ldots,w_n) \propto \log p(\text{Spam}) \prod_{i=1}^{n} p(w_i|\text{Spam}) \quad (1)$$

$$\log p(\text{Spam}|w_1,\ldots,w_n) \propto \log p(\text{Spam}) + \log \prod_{i=1}^{n} p(w_i|\text{Spam}) \quad (2)$$

$$\log p(\text{Spam}|w_1,\ldots,w_n) \propto \log p(\text{Spam}) + \sum_{i=1}^{n} \log p(w_i|\text{Spam}) \quad (3)$$

Experiment Content:

1. Data Loading: Use the Enron email dataset, which includes real email data from Enron after its collapse. The dataset contains both spam and normal emails.



2. Naive Bayes Classifier: Implement a spam email classifier based on the Naive Bayes algorithm. The classifier needs to learn prior probabilities, a vocabulary list, and the frequency of

each word in spam and normal emails.

```
def get_data(DATA_DIR):
    subfolders = ['enron%d' % i for i in range(1,7)]
    data = []
    target = []
    for subfolder in subfolders:
        # spam
        spam_files = os.listdir(os.path.join(DATA_DIR, subfolder, 'spam'))
        for spam_file in spam_files:
            with open(os.path.join(DATA_DIR, subfolder, 'spam', spam_file), encoding="latin-1") as f:
                data.append(f.read())
                target.append(1)
        # ham
        ham_files = os.listdir(os.path.join(DATA_DIR, subfolder, 'ham'))
        for ham_file in ham_files:
            with open(os.path.join(DATA_DIR, subfolder, 'ham', ham_file), encoding="latin-1") as f:
                data.append(f.read())
                target.append(0)
    return data, target
```

3. Data Processing: Cleanse text data by removing punctuation and converting the text into a sequence of words.

```
class SpamDetector_1(object):
    """Implementation of Naive Bayes for binary classification"""
    def clean(self, s):
        translator = str.maketrans("", "", string.punctuation)
        return s.translate(translator)
    def tokenize(self, text):
        text = self.clean(text).lower()
        return re.split("\W+", text)
    def get_word_counts(self, words):
        word_counts = {}
        for word in words:
            word_counts[word] = word_counts.get(word, 0.0) + 1.0
        return word_counts
```

4. Model Training: Train the Naive Bayes classifier using the training set, calculating prior probabilities and word frequencies.

```
class SpamDetector_2(SpamDetector_1):
    def fit(self, X, Y):
        self.num_messages = {}      # 用于存储邮件总数，其中分为 spam 和 ham
        self.log_class_priors = {}  # 用于存储对数先验概率，其中分为 spam 和 ham
        self.word_counts = {}       # 用于存储单词在垃圾邮件和正常邮件中出现次数
        self.vocab = set()          # 全局词汇表
        n = len(X)
        self.num_messages['spam'] = sum(1 for label in Y if label == 1)
        self.num_messages['ham'] = sum(1 for label in Y if label == 0)

        # X Y 分别对应之前的 data 和 target

        # Compute the log prior probability of spam/ham.
        # Hint: compute the log class priors by counting up how many spam/ham messages
        # are in our dataset and dividing by the total number, and take the log.
        # Please perform the calculation for self.log_class_priors['spam'] and self.log_class_priors['ham'] separately.

        self.log_class_priors['spam'] = math.log(self.num_messages['spam'] / n)
        self.log_class_priors['ham'] = math.log(self.num_messages['ham'] / n)

        self.word_counts['spam'] = {}
        self.word_counts['ham'] = {}

        for x, y in zip(X, Y):
            c = 'spam' if y == 1 else 'ham'

            counts = self.get_word_counts(self.tokenize(x))

            for word, count in counts.items():
                if word not in self.vocab:
                    self.vocab.add(word)
                if word not in self.word_counts[c]:
                    self.word_counts[c][word] = 0.0
                self.word_counts[c][word] += count      # 计数
```

5. Model Prediction: Use the test set to predict with the trained model, determining whether each email is spam or normal.

```
def predict(self, X):
    result = []
    flag_1 = 0
    for x in X:
        counts = self.get_word_counts(self.tokenize(x))
        spam_score = 0
        ham_score = 0
        flag_2 = 0
        for word, _ in counts.items():
            if word not in self.vocab: continue

            # According to Equation 3, compute the conditional probability of spam/ham and add Laplace
            # smoothing (add 1 to the numerator and add the size of the vocabulary to the denominator).
            # Please define the variable name as log_w_given_spam and log_w_given_ham.

            log_w_given_spam = math.log((self.word_counts['spam'].get(word, 0.0) + 1) / (sum(self.word_counts['spam'].values()) + len(self.vocab
            log_w_given_ham = math.log((self.word_counts['ham'].get(word, 0.0) + 1) / (sum(self.word_counts['ham'].values()) + len(self.vocab)))


            # Testing your results. Do not remove.
            if (flag_1 == 0) and (flag_2 == 0):
                # If an assert error occurs, you can print and debug it.
                print("log_w_given_spam", log_w_given_spam)
                print("log_w_given_ham", log_w_given_ham)

                assert round(log_w_given_spam, 4) == -5.2759, "The conditional probability of spam is calculated incorrectly, please try again."
                assert round(log_w_given_ham, 4) == -5.1075, "The conditional probability of ham is calculated incorrectly, please try again."

            # Calculate the sum of the conditional probabilities of spam/ham and define them as spam_score and ham_score.

            spam_score += log_w_given_spam
            ham_score += log_w_given_ham

            flag_2 += 1
```

Experiment Process：

1. Data Loading: Load the Enron email dataset using the provided code and divide the data into training and test sets.

```
X, y = get_data(DATA_DIR)

assert len(X) == 33716, "Please check for missing files."
assert len(y) == 33716, "Please check for missing files."

# Print the first data and target.
print(X[0])
print(y[0])
```

✓ 25.1s

```
Subject: dobmeos with hgh my energy level has gone up ! stukm
introducing
```

2. Model Training: Train the Naive Bayes classifier using the

training set, calculating prior probabilities and word frequencies.

```
MNB = SpamDetector_2()
MNB.fit(X[100:], y[100:])

# If an assert error occurs, you can print and debug it.
print("log_class_priors of spam", MNB.log_class_priors['spam'])
print("log_class_priors of ham", MNB.log_class_priors['ham'])

assert round(MNB.log_class_priors['spam'], 4) == -0.6776, "The prior probability of spam is calculated incorrectly, please try again."
assert round(MNB.log_class_priors['ham'], 4) == -0.7089, "The prior probability of ham is calculated incorrectly, please try again."
```

✓ 3.8s

```
log_class_priors of spam -0.6776210267040288
log_class_priors of ham -0.7089182027290691
```

3. Model Prediction: Use the test set to predict with the trained model and obtain classification results.

```
MNB = SpamDetector()
MNB.fit(X[100:], y[100:])
pred = MNB.predict(X[:100])
true = y[:100]

# Compute the accuracy rate and print it (results are retained to 4 decimal places).
# Please define the variable name as accuracy.

accuracy = sum(1 for p, t in zip(pred, true) if p == t) / len(true)

print(accuracy)
assert accuracy == 0.9800
```

✓ 28.7s

```
log_w_given_spam -5.275940294514414
log_w_given_ham -5.107477914537173
spam_score -1015.1113024344844
ham_score -1102.456441377472
spam_score plus prior_probability -1015.7889234611885
ham_score plus prior_probability -1103.165359580201
0.98
```

Data Logging and Processing:

Use the provided code to load the Enron email dataset, dividing the data into spam and normal emails. Process text data by removing punctuation and converting the text into a sequence of words.

采用class定义体现了面向对象编程，有助于组织代码。

clean 清除邮件中的标点 *string.punctuation包含所有标点。*

tokenize 将大写转化为小写 *\W+ 识别非单词字符，即空格。从而实现单词之间的分割。*

get_word_counts 单词计数 *利用 tokenize 的结果*

```
class SpamDetector_1(object):
    """Implementation of Naive Bayes for binary classification"""
    def clean(self, s):
        translator = str.maketrans("", "", string.punctuation)
        return s.translate(translator)
    def tokenize(self, text):
        text = self.clean(text).lower()
        return re.split("\W+", text)
    def get_word_counts(self, words):
        word_counts = {}
        for word in words:
            word_counts[word] = word_counts.get(word, 0.0) + 1.0
        return word_counts
```

Experimental Results and Analysis:

The well-trained model is used to classify the test set, and the classification accuracy is calculated.

The experimental results indicate that the trained Naive Bayes spam email classifier achieved a high accuracy on the test data, reaching 0.98. This means that the model successfully identified the majority of spam emails and performed well on the test set.

```
MNB = SpamDetector()
MNB.fit(X[100:], y[100:])
pred = MNB.predict(X[:100])
true = y[:100]

# Compute the accuracy rate and print it (results are retained to 4 decimal places)
# Please define the variable name as accuracy.

accuracy = sum(1 for p, t in zip(pred, true) if p == t) / len(true)

print(accuracy)
assert accuracy == 0.9800
```

✓ 32.5s

```
log_w_given_spam -5.275940294514414
log_w_given_ham -5.107477914537173
spam_score -1015.1113024344844
ham_score -1102.456441377472
spam_score plus prior_probability -1015.7889234611885
ham_score plus prior_probability -1103.165359580201
0.98
```

指导教师批阅意见：   成绩评定：   指导教师签字： 年　　月　　日
备注：