计算机学院 社交媒体与舆情分析 课程实验报告

实验题目：1 Sentiment Analysis		学号：201800130112
日期：2021.10.20	班级： 计科 18.3	姓名： 赵子涵
Email：zih_an@163.com		

实验方法介绍：

一、实验内容简介

- 方法及目标：使用朴素贝叶斯（Naïve Bayes）做情感分析（Sentiment Analysis）。
- 数据：文件分为 train 和 test 目录，每个目录下都有 pos 和 neg 目录代表情感正负。其中，每个文件都是一篇短文。

二、算法介绍

1. Tokenization 分词

本实验基于 unigram 模型的方式分词，即每个词对应一个位置。

e.g. I am a student. => [I, am, a, student]

2. Feature Extraction -- TF-IDF

TF-IDF：Term Frequency - Inverse Document Frequency。这是用于衡量在一个文档集合中，一个词在某一篇文档中的重要性。分为两项，TF 和 IDF 综合得到：

TF-IDF = TF * IDF

$$TF_w = \frac{\text{在某一类中词条}w\text{出现的次数}}{\text{该类中所有的词条数目}}$$

$$IDF = log(\frac{\text{语料库的文档总数}}{\text{包含词条}w\text{的文档数}+1}),$$

3. Classification -- Naïve Bayes

设数据集为 X，类型为 y，欲求使得 P(y|X)最大的 y 值作为类别：

目标：$argmax_y\ P(y|X)$，需要对其进行估计。

- $P(y|X) = \frac{P(y)P(X|y)}{P(X)} \propto P(y)P(X|y) = P(y)\prod_i P(X^{(i)}|y)$

使用 log 处理，乘法计算改为加法：

$$\log P(y)\prod_i P(X^{(i)}|y) = logP(y) + \sum_i \log P(X^{(i)}|y)$$

当 X 具有多个特征时，

$$\log P(y)\prod_{i,j}P\left(X_j^{(i)}\middle|y\right)=logP(y)+\sum_{i,j}\log P\left(X_j^{(i)}\middle|y\right)$$

由此转为分别对$P(y)$和$P\left(X_j^{(i)}\middle|y\right)$进行估计，对于二者的解本质是由 MLE 推导求得，结果如下：

$$p(t\mid y)=\frac{\sum_{i=1}^{m}\mathbf{1}(y^{(i)}=y)count^{(i)}(t)}{\sum_{i=1}^{m}\mathbf{1}(y^{(i)}=y)\sum_{t=1}^{v}count^{(i)}(t)}$$

$$p(y)=\frac{\sum_{i=1}^{m}\mathbf{1}(y^{(i)}=y)}{m}$$

$$\text{where}\quad count^{(i)}(t)=\sum_{j=1}^{n_i}\mathbf{1}(x_j^{(i)}=t)$$

为防止除 0 或为 0 的现象出现，使用拉普拉斯平滑（Laplace Smoothing）处理分子分母：

Laplace smoothing

$$\psi(t\mid y)=\frac{\sum_{i=1}^{m}\mathbf{1}(y^{(i)}=y)count^{(i)}(t)+1}{\sum_{i=1}^{m}\mathbf{1}(y^{(i)}=y)\sum_{t=1}^{v}count^{(i)}(t)+v}$$

$$\psi(y)=\frac{\sum_{i=1}^{m}\mathbf{1}(y^{(i)}=y)+1}{m+k}$$

三、将 TFIDF 处理的数据集用于 Naïve Bayes

在一般的朴素贝叶斯中计数，默认是每个特征/数据条目算 1 个，TFIDF 计算得到的整数值可以作为代替这个 1 的权重，即：

$$P\left(X^{(i)}|y\right)=\frac{count\left(X^{(i)},y\right)+1}{count(y)+|V|}=\frac{\sum_i TFIDF\ of\ ith\ X\ with\ label\ y}{sum\ of\ all\ TFIDF\ of\ X\ with\ label\ y}=\frac{[vector]}{[constant]}$$

在预测时，原本根据有无（1/0）求和对应条件概率的部分，也改为 TFIDF 值直接与该权重相乘。

实验过程描述：（不要求罗列完整源代码）

一、文本处理

- 加载数据：

```
def loadData(flagTrain = True):
    path = './aclImdb/'
    if flagTrain:
        path += "train/"
    else:
        path += "test/"
```

```
pos_path = path + 'pos/'
neg_path = path + 'neg/'
pos_files = [pos_path + x for x in
                    filter(lambda x: x.endswith('.txt'), os.listdir(pos_path))]
neg_files = [neg_path + x for x in
                    filter(lambda x: x.endswith('.txt'), os.listdir(neg_path))]
pos_list = [open(x, 'r', encoding='utf-8').read().lower() for x in pos_files]
neg_list = [open(x, 'r', encoding='utf-8').read().lower() for x in neg_files]
data_list = pos_list + neg_list
label_list = [1] * len(pos_list) + [0] * len(neg_list)

# shuffle if you'd like ============================
if flagTrain:
    merged_data = list(zip(data_list, label_list))
    shuffle(merged_data)
    data_list, label_list = list(zip(*merged_data))
return list(data_list), list(label_list)
```

- 分词与预处理：

```
from keras.preprocessing.text import Tokenizer
max_vocab_size = 50000
tokenizer = Tokenizer(num_words=max_vocab_size, oov_token='<UNK>')
tokenizer.fit_on_texts(data_list)
tf_idf_data = tokenizer.texts_to_matrix(data_list, mode='tfidf')
```

二、朴素贝叶斯

In [22]:
```
class NaiveBayes():
    def fit(self, X, y):
        self.num_classes = 2  # neg/pos = 0/1
        self.m_examples = y.shape[0]
        ## p(X/y)
        self.prob_Xy_arr = np.zeros((self.num_classes, X.shape[1]), dtype=np.float64)
        count_y = np.zeros((self.num_classes, 1))
        for i in range(self.m_examples):
            ith_lbl = y[i]
            self.prob_Xy_arr[ith_lbl] += X[i]
            count_y[ith_lbl] += np.sum(X[i])
        self.prob_Xy_arr =  (self.prob_Xy_arr + 1) / (count_y + X.shape[1])

        ## p(y)
        self.prob_y_arr = np.zeros(self.num_classes, dtype=np.float64)
        for i in range(self.num_classes):
            self.prob_y_arr[i] = sum(y==i) / self.m_examples

    def predict(self, X):
        m_test = X.shape[0]
        labels = np.zeros(m_test)
        for i in range(m_test):
            y, prob = None, float('-inf')
            for lbl in range(self.num_classes):
                sc = np.sum(X[i] * np.log(self.prob_Xy_arr[lbl]) + np.log(self.prob_y_arr[lbl]))  # X is one-hot encoding
                if sc > prob:
                    prob = sc
                    y = lbl
            labels[i] = y
        return labels
```

In [23]:
```
nb = NaiveBayes()
nb.fit(tf_idf_data, label_list)
```

三、训练与预测，并使用 scikit-learn 库的朴素贝叶斯预测

- 训练与预测

```
testdata_tfidf = tokenizer.texts_to_matrix(testdata, mode='tfidf')
testlabel = np.array(testlabel)
print("test label shape: ", testlabel.shape)
print(testlabel)
print("test data shape: ", testdata_tfidf.shape)
print(testdata_tfidf)
```

```
predlabels = nb.predict(testdata_tfidf)
print(predlabels)
acc = predlabels==testlabel
print("accuracy: ", np.sum(acc) / testlabel.shape[0])
```

```
[1. 1. 1. ... 1. 0. 1.]
accuracy:  0.78516
```

- 使用 scikit-learn

使用 tensorflow 预处理结果（即本实验使用的数据），sklearn 的朴素贝叶斯，结果如下：

```
tf_vectorizer = CountVectorizer() # or term frequency
X_train_tf = tf_vectorizer.fit_transform(data_list)
X_test_tf = tf_vectorizer.transform(testdata)

naive_bayes_classifier = MultinomialNB()
# naive_bayes_classifier.fit(X_train_tf, label_list)
# y_pred = naive_bayes_classifier.predict(X_test_tf)

naive_bayes_classifier.fit(tf_idf_data, label_list)
y_pred = naive_bayes_classifier.predict(testdata_onehot)

score1 = metrics.accuracy_score(testlabel, y_pred)
print("accuracy:   %0.3f" % score1)
```

```
accuracy:   0.785
```

纯粹使用 scikit-learn 的 CountVectorizer+朴素贝叶斯，结果如下：

In [12]:
```
from time import time
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
```

In [13]:
```
tf_vectorizer = CountVectorizer() # or term frequency
X_train_tf = tf_vectorizer.fit_transform(data_list)
X_test_tf = tf_vectorizer.transform(testdata)

naive_bayes_classifier = MultinomialNB()
naive_bayes_classifier.fit(X_train_tf, label_list)
y_pred = naive_bayes_classifier.predict(X_test_tf)

score1 = metrics.accuracy_score(testlabel, y_pred)
print("accuracy:   %0.3f" % score1)
```

```
accuracy:   0.814
```

纯粹使用 scikit-learn 的 TfidfVectorizer+朴素贝叶斯，结果如下：

```
:  tf_vectorizer = TfidfVectorizer() # or term frequency
   X_train_tf = tf_vectorizer.fit_transform(data_list)
   X_test_tf = tf_vectorizer.transform(testdata)

   naive_bayes_classifier = MultinomialNB()
   naive_bayes_classifier.fit(X_train_tf, label_list)
   y_pred = naive_bayes_classifier.predict(X_test_tf)

   # naive_bayes_classifier.fit(tf_idf_data, label_list)
   # y_pred = naive_bayes_classifier.predict(testdata_onehot)

   score1 = metrics.accuracy_score(testlabel, y_pred)
   print("accuracy:   %0.3f" % score1)
```

```
accuracy:   0.830
```

结论分析：

使用相同数据及处理发现，准确率较高，与 scikit-learn 结果相同，手写的贝叶斯分类器正确。

但使用 sklearn 的 tfidf 数据预处理时，结果有部分差异，认为与 sklearn 内置的 tfidf 处理细节及分词情况与 tensorflow 不同导致的，。

但是，该准确率还不够高，仅 0.785 左右，猜测与分词有关。本实验中，分词使用 unigram 的方式。考虑到数据是文档，因此合理假设使用 bigram，trigram，N-gram 的方式准确率应该会更高，需要进一步实验验证。

结论：

基于 unigram 模型，使用朴素贝叶斯（Naïve Bayes）与 TF-IDF 能很好的分类文本数据，也即是做情感分析的 positive/negative 二分类问题。

核心代码——朴素贝叶斯类：

```
class NaiveBayes():
    def fit(self, X, y):
        self.num_classes = 2    # neg/pos = 0/1
        self.m_examples = y.shape[0]
        ## p(X|y)
        self.prob_Xy_arr = np.zeros((self.num_classes, X.shape[1]), dtype=np.float64)
        count_y = np.zeros((self.num_classes, 1))
        for i in range(self.m_examples):
            ith_lbl = y[i]
            self.prob_Xy_arr[ith_lbl] += X[i]
            count_y[ith_lbl] += np.sum(X[i])
        self.prob_Xy_arr =    (self.prob_Xy_arr + 1) / (count_y + X.shape[1])

        ## p(y)
```

```
        self.prob_y_arr = np.zeros(self.num_classes, dtype=np.float64)
        for i in range(self.num_classes):
            self.prob_y_arr[i] = sum(y==i) / self.m_examples

    def predict(self, X):
        m_test = X.shape[0]
        labels = np.zeros(m_test)
        for i in range(m_test):
            y, prob = None, float('-inf')
            for lbl in range(self.num_classes):
                sc = np.sum(X[i] * np.log(self.prob_Xy_arr[lbl]) + np.log(self.prob_y_arr[lbl]))
                if sc > prob:
                    prob = sc
                    y = lbl
            labels[i] = y
        return labels
```