人工智能实验——电影评价分类器

一、实验内容

Large Movie Review Dataset 电影评价数据集，是一个用于情感二分类的数据集。从训练数据 traindata 和其对应的标号 trainlabel 中学习相应的分类模型。在测试过程中，用学习到的模型对测试集中的数据 testdata 作预测，并将预测结果与测试数据的真实标签 testlabel 进行比较，从而度量分类模型的性能。

二、实验要求

1. 实现数据集的特征抽取(10%)

提 交一个 Python 函数 getFeature(comment)，其中 comment 为一条评价，要

求函数返回从一条评价中抽取的特征向量。特征向量抽取的方法为 bag of words，但由于词典库过于庞大，需要对该特征进行优化。思考优化方法，并在实验报告中给出选择的原因与分析过程。

2 . 实现一个朴素贝叶斯分类器(10%)

提交一个 Python 函数 nBayesClassifier(traindata, trainlabel,testdata, testlabel, threshold)，其中 threshold 为用于判断类别的后验概率的阈值，即如果 P(good|comment) > threshold 则判断为正面评价。要求函数返回对测试数据的预测 ypred，以及通过与 ground truth(真实评价),比较计算得到的分类正确率 accuracy，ypred 与 accuracy以 tuple 形式返回。

3. 实现一个最小二乘分类器(引入规范化项后)(10%)

1). 对引入了 L2 规范化项之后的最小二乘分类问题进行推导。

2). 基于 1 中的结果，实现并提交一个 Python 函数 lsClassifier(traindata,trainlabel, testdata, testlabel, lambda) 4. 实现一个支持向量机分类器 (15 %)

提交一个 Python 函数 softsvm(traindata, trainlabel, testdata,testlabel, sigma, C)，其中 C 为 soft margin SVM的控制参数，sigma 为控制核函数的参数，当 sigma=0 时，使用线性核函数，其他情况则使用 RBF 核函数。

5. 在不同数据集上使用交叉验证选择各个算法的参数(15%)

实现交叉验证（代码需要提交），在各个数据集上：使用 5 -fold 交叉验证为每个算法挑选适当的参数(Naïve Bayes 中的 threshold，最小二乘法中的 L ambda，SVM 中的 sigma 和 C）；对每一个算法：返回一个矩阵，表示每一个参数（参数组合）在每一个 fold 上的正

确率（若有 10 个参数，则返回 10x5 的矩阵）；挑选在 5 个 fold 中平均正确率最高的参数（参数组合）在实验报告中需要记录交叉验证的结果，即对于每个参数(参数组合)在 5 个

fold 上的平均正确率。

三、实验过程

1. 实现数据集的特征抽取

采用 tf-idf 方法。TF-IDF 是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。TF－IDF 的主要思想是：如果某个词或短语在一篇文章中出现的频率 TF 高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。TFIDF 实际上是：TF * IDF，TF 词频(Term Frequency)，IDF 逆向文件频率(Inverse Document Frequency)。TF 表示词条在文档 d 中出现的频率。**词频**（term frequency，TF）指的是某一个给定的词语在该文件中出现的频率。这个数字是对**词数**(term count)的归一化，以防止它偏向长的文件。（同一个词语在长文件里可能会比短文件有更高的词数，而不管该词语重要与否。）对于在某一特定文件里的词语来说，它的重要性可表示为：

$$\mathrm{tf}_{\mathrm{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

以上式子中分子是该词在文件中的出现次数，而分母则是在文件中所有字词的出现次数之和。

逆向文件频率（inverse document frequency，IDF）是一个词语普遍重要性的度量。某一特定词语的 IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到：

$$\mathrm{idf}_{\mathrm{i}} = \log \frac{|D|}{|\{j : t_i \in d_j\}|}$$

|D|：语料库中的文件总数

然后再计算 TF 与 IDF 的乘积。

$$\mathrm{tfidf}_{i,j} = \mathrm{tf}_{i,j} \times \mathrm{idf}_i$$

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的

TF-IDF。因此，TF-IDF 倾向于过滤掉常见的词语，保留重要的词语。

对于本实验，在抽取每一条评价的特征向量时可以对其作归一化处理 tf 值，而 idf 值需要事先计算好。

对数据作预处理，idf.txt 文件：

Bagofword.py:

```
from scipy import sparse
import numpy as np
import csv
from sklearn import preprocessing
word=np.zeros(89527)
toatal=np.ones(89527)
toatal=toatal*50000
with open("data","r") as file:
    for line in file.readlines():
        line = line.strip().split(' ')
        v = getFeature(line)
        word = word+np.where(v!=0, 1, 0)
```

4．实现一个支持向量机分类器

```
idf = np.log(np.true_divide(toatal,word))
out = open("idf.txt", 'w')
csv_writer = csv.writer(out)
csv_writer.writerow(idf)
```

所以，最终的 getFeature 函数为：

```
def getFeature(comment): # read a comment
    row=[]
    col=[]
    data=[]
    comment=comment[1:]
    idf = np.loadtxt(open("idf.txt", "r"), delimiter=",")
    for i in range(len(comment)):
        index_value=comment[i].split(":")
        index=int(index_value[0])
        value = int(index_value[1])
        col.append(index)
        data.append(value)
        row.append(0)
    vcom = sparse.coo_matrix((data, (row, col)), shape=(1, 89527))
    v = np.array(vcom.toarray())
    v= preprocessing.normalize(v, norm='l1')
```

```
    v=v[0]*idf
    return v
```

注：预处理过程中的 getFeature 与之类似，只不过没有归一化和乘以 idf 过程，所以不再列出。

2. 实现一个朴素贝叶斯分类器

调用 scikity－learning 库中的多项式朴素贝叶斯分类器

```
from sklearn.naive_bayes import MultinomialNB

#threshold=0.5,0.6,0.7,0.75,0.8,0.9
def nBayesClassifier(traindata,trainlabel,testdata,testlabel,threshold):
    clf = MultinomialNB()
    clf.fit(traindata,trainlabel)
    ypred= clf.predict_proba(testdata)
    ypred=np.where(ypred[:,1]>threshold,1,-1)
    count=0
    for i in range(len(testlabel)):
        if testlabel[i]==ypred[i]:
            count=count+1
    accuracy=float(count)/len(testlabel)
    return ypred,accuracy
```

3. 实现一个最小二乘分类器(引入规范化项后)

调用 scikity－learning 库中的 Ridge Regression 分类器

```
from sklearn import linear_model

#lambda=1e-4,0.01,0.1,1,0.5,1,10,100,1000,5000,10000
def lsClassifier(traindata,trainlabel,testdata,testlabel,Lambda):
    reg = linear_model.Ridge(alpha=Lambda)
    reg.fit(traindata,trainlabel)
    ypred=reg.predict(testdata)
    ypred = np.where(ypred > 0, 1, -1)
    count = 0
    for i in range(len(testlabel)):
        if testlabel[i] == ypred[i]:
            count = count + 1
    accuracy = float(count) / len(testlabel)
    return ypred, accuracy
```

4. 实现一个支持向量机分类器

调用 scikity－learning 库中的支持向量机分类器

```
from sklearn import svm

#singma=0.01d,0.1d,d,10d,100d
#c=1,10,100,1000
def softsvm(traindata,trainlabel,testdata,testlabel,sigma,c):
    if sigma==0:
        clf = svm.SVC(kernel='linear',C=c)
    else:
        clf=svm.SVC(kernel='rbf',gamma=1.0/(sigma*sigma),C=c)
    clf.fit(traindata, trainlabel)
    ypred=clf.predict(testdata)
    count = 0
    for i in range(len(testlabel)):
        if testlabel[i] == ypred[i]:
            count = count + 1
    accuracy = float(count) / len(testlabel)
    return ypred,accuracy
```

5. 在不同数据集上使用交叉验证选择各个算法的参数

调用 scikity－learning 库中的 kflod 实现交叉验证:

```
from sklearn.model_selection import KFold

kf=KFold(n_splits=5,shuffle=True)
threshold=[0.5,0.6,0.7,0.75,0.8,0.9]
Lambda=[0.0001,0.01,0.1,0.5,1,10,100,1000,5000,10000]
nbasy_accuracy=np.zeros((len(threshold),5))
lsq_accuracy=np.zeros((len(Lambda),5))

for i in range(len(threshold)):
    j=0
    for train_index, test_index in kf.split(traindata, trainlabel):
        ypred, nbasy_accuracy[i,j] =
nBayesClassifier(traindata[train_index],trainlabel[train_index],
traindata[test_index],trainlabel[test_index], threshold[i])
        j=j+1
print nbasy_accuracy
np.savetxt("nbasy_accuracy.txt",nbasy_accuracy)

for i in range(len(Lambda)):
    j = 0
    for train_index, test_index in kf.split(traindata, trainlabel):
        ypred, lsq_accuracy[i,j] = lsClassifier(traindata[train_index],
trainlabel[train_index], traindata[test_index],trainlabel[test_index], Lambda[i])
        print(lsq_accuracy[i,j])
        j=j+1
```

```
print lsq_accuracy
np.savetxt("lsq_accuracy.txt",lsq_accuracy)
```

6、支持向量机 d 参数的计算。

由于数据集矩阵的大小是 50000*85279，若直接计算 d 大小会耗费大量时间。采用随机抽样的办法计算 d 可以作为近似，发现 d 的大小稳定在 0.033（不同的特征抽取方法会用有不同的结）。

计算 d 的代码：

```
from scipy import sparse
import numpy as np
from sklearn import preprocessing
import linecache
import random

count=0
n=1000
row = []
col = []
data = []
idf= np.loadtxt(open("idf.txt","r"),delimiter=",")
for i in range(n):
    a = random.randrange(0, 50000)
    line = linecache.getline('data', a)
    line = line.strip().split(' ')
    comment = line[1:]
    for i in range(len(comment)):
        index_value = comment[i].split(":")
        index = int(index_value[0])
        value = int(index_value[1])
        col.append(index)
        data.append(value * idf[index])
        row.append(count)
    count = count + 1
testdata = sparse.coo_matrix((data, (row, col)), shape=(n, 89527))
testdata=preprocessing.normalize(testdata, norm='l1')
testdata=np.array(testdata.todense())
result=0
for i in range(n):
    for j in range(n):
        if i!=j:
            result += np.dot((testdata[i] - testdata[j]) , (testdata[i] -
testdata[j]))
    #print(i,result)
result=result/(n*n)
print(result)
```

四、实验结果

朴素贝叶斯分类器在 5 个 fold 上的结果：

Threshold:	Accuracy:(5 fold)					accuracy average:
0.5:	[0.8659	0.8756	0.8601	0.8649	0.8706]	0.86742
0.6	[0.4959	0.5042	0.498	0.4964	0.5061]	0.50012
0.7	[0.5023	0.4955	0.5034	0.5018	0.497]	0.5
0.75	[0.5024	0.4974	0.4971	0.4951	0.508]	0.5
0.8	[0.5004	0.4986	0.5018	0.4987	0.5005]	0.5
0.9	[0.5005	0.5003	0.5039	0.5011	0.4942]	0.5

从中可以很明显的看出 threshold=0.5 分类的效果最好，原因在于 P(good|comment) + P（bad|comment)=1;测试数据在分类之后得到的概率值都是在 0.5 附近的，部分正面评价略大于 0.5，部分负面评价略小于 0.5。

引入规范化最小二乘分类器在 5 个 fold 上的结果：

Lamda:	Accuracy:(5 fold)					accuracy average:
0.0001	[0.8675	0.8671	0.866	0.8692	0.866]	0.86716
0.01	[0.8932	0.8884	0.8952	0.8907	0.8908]	0.89166
0.1	[0.8951	0.8996	0.8981	0.8966	0.9031]	0.8985
0.5	[0.8837	0.8786	0.8845	0.8872	0.88]	0.8828
1	[0.8738	0.8665	0.8708	0.8733	0.875]	0.87188
10	[0.8389	0.8282	0.841	0.8418	0.8388]	0.83774
100	[0.8353	0.7881	0.8352	0.8183	0.8077]	0.81692
1000	[0.5021	0.6709	0.6176	0.4989	0.6669]	0.59128
5000	[0.4978	0.4932	0.4977	0.4978	0.4909]	0.49548
10000	[0.5012	0.498	0.4957	0.4922	0.5004]	0.4975

从中可以看出 lamda = 0. 1 时分类效果最好，分析原因是因为防止过拟合而设置的正则化惩罚不宜过大也不宜过小。

由于支持向量机进行一次运算的时间太长（大概每次 10-15 分钟），只计算出部分结果：

Sigma=0.33,c=1：Accuracy=0.90001

Sigma=0.33,c=10: accuracy=0.8982
Sigma=3.3,c=1:Accuracy=0.4939
Sigma=0.00033,c=1: accuracy=0.5146

初步结论，参数选为 sigma=0.33,c=1 较好。

总结，朴素贝叶斯分类器和最小二乘法分类器在运行时间上较快，选用正确的参数能得到较好的结果。支持向量机调整好参数能得到更高的准确率，但运行时间太慢了。