第1题 tf-idf向量表示和文本分类（30分）

In [2]:

```
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.feature_extraction.text import TfidfTransformer  #权重计算
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer  #词频统计
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [2]:

```
#review corpus: [reveiws,y]
y=[]
reviews=[]
with open('train.txt','r',encoding='utf-8') as infile: # train.txt=>sentence list
    for line in infile:
        label, sentence = line.strip().split("\t")
        y.append(int(label))
        reviews.append(sentence)


x_train,x_test,y_train,y_test=train_test_split(reviews, y,test_size=0.2)
```

In [3]:

```
###编写代码，实现用tf-idf对语料x_train,x_test进行向量表示,输出向量矩阵train_vecs,test_vecs


#该类会将文本中的词语转换为词频矩阵，矩阵元素a[i][j] 表示j词在i类文本下的词频
from cgi import test


vectorizer = CountVectorizer()
#该类会统计每个词语的tf-idf权值
tf_idf_transformer = TfidfTransformer()
#将文本转为词频矩阵并计算tf-idf
train_vecs = tf_idf_transformer.fit_transform(vectorizer.fit_transform(x_train)).toarray()


#对测试集进行tf-idf权重计算
test_vecs = tf_idf_transformer.transform(vectorizer.transform(x_test)).toarray()


print(train_vecs,test_vecs)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]] [[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

In [4]:

```
from sklearn.preprocessing import scale
train_vecs=scale(train_vecs)
test_vecs=scale(test_vecs)

#from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report
from sklearn import  svm

clf = svm.SVC(kernel='linear', gamma=10)
clf.fit(train_vecs,y_train)
y_pred=clf.predict(test_vecs)
print(classification_report(y_test,y_pred))
print('Acc: %.2f' % clf.score(test_vecs, y_test))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       610
           1       0.99      1.00      0.99       808

    accuracy                           0.99      1418
   macro avg       0.99      0.99      0.99      1418
weighted avg       0.99      0.99      0.99      1418

Acc: 0.99
```

In [5]:

```

#using SLG to train classifier model, and assess
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import classification_report
lr=SGDClassifier(loss='log',penalty='l1')
lr.fit(train_vecs,y_train)
y_pred = lr.predict(test_vecs)
print(classification_report(y_test,y_pred))

print('Acc: %.2f' % lr.score(test_vecs, y_test))
```

```
c:\Python\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:173: Future
Warning: The loss 'log' was deprecated in v1.1 and will be removed in version 1.3. U
se `loss='log_loss'` which is equivalent.
  warnings.warn(

              precision    recall  f1-score   support

           0       0.93      0.97      0.95       610
           1       0.97      0.95      0.96       808

    accuracy                           0.95      1418
   macro avg       0.95      0.96      0.95      1418
weighted avg       0.95      0.95      0.95      1418

Acc: 0.95
```

第2题 word2vec向量表示和文本分类（30分）

In [1]:

```
from sklearn.model_selection import train_test_split
import word2vec    #w2v包
import numpy as np

#review corpus: [reveiws,y]
y=[]
reviews=[]
with open("train.txt",'r',encoding='utf-8') as infile: # train.txt=>sentence list
    for line in infile:
        label, sentence = line.strip().split("\t")
        y.append(int(label))
        reviews.append(sentence)

x_train,x_test,y_train,y_test=train_test_split(reviews, y,test_size=0.2)


#sentence ->word list
def cleanText(corpus):  #输入corpus: sentence list
    corpus=[z.lower().replace('\n','').split() for z in corpus]
    # corpus=[z.replace('.','').split() for z in corpus]

    return corpus

x_train=cleanText(x_train)   # =>lowercase, word list
x_test=cleanText(x_test)


def claenagain(corpus):
    for wordlist in corpus:
        for word in wordlist:
            word = ''.join(filter(str.isalpha, word))
    return corpus


x_train=claenagain(x_train)
x_test=claenagain(x_test)
```

In [2]:

```
###编写代码，采用自训练或者预训练方式获得word2vec词向量

from operator import mod
from gensim.models import KeyedVectors
model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
from gensim.test.utils import datapath, get_tmpfile   #word similarity package
from gensim.scripts.glove2word2vec import glove2word2vec

eg = model['fuck']
print(eg)
```

```
[ 1.04492188e-01 -3.12500000e-01  2.13867188e-01  3.08593750e-01
 -1.60156250e-01  9.42382812e-02 -1.60156250e-01 -3.22265625e-02
  1.69921875e-01  8.49609375e-02 -1.38671875e-01 -2.31445312e-01
 -6.49414062e-02  6.68945312e-02 -2.16796875e-01  1.82617188e-01
 -2.17773438e-01 -8.78906250e-03 -7.32421875e-02 -2.44140625e-01
  1.52343750e-01  1.20605469e-01  3.47656250e-01  7.42187500e-02
  7.03125000e-02  2.05078125e-01 -1.24023438e-01  1.80664062e-01
  1.02539062e-01 -1.17675781e-01  9.08203125e-02  9.71679688e-02
 -1.92871094e-02  1.95312500e-01 -1.08886719e-01  6.49414062e-02
  3.32031250e-01 -7.37304688e-02 -1.27929688e-01  4.96093750e-01
  1.16699219e-01 -6.54296875e-02  3.06640625e-01  1.28906250e-01
 -3.27148438e-02 -1.74804688e-01 -1.33789062e-01 -2.79296875e-01
 -1.66992188e-01  1.60156250e-01 -3.16406250e-01 -1.95312500e-02
  1.73339844e-02  2.63671875e-01  9.76562500e-02 -1.37939453e-02
 -1.36718750e-01 -2.14843750e-01  2.07519531e-02 -1.17675781e-01
  7.03125000e-02 -1.10351562e-01  3.39355469e-02 -6.03027344e-02
  1.74560547e-02 -4.04296875e-01 -2.10937500e-01  1.75781250e-02
 -2.18750000e-01  1.01562500e-01  1.29882812e-01  1.99218750e-01
  8.36181641e-03  1.27929688e-01 -4.51171875e-01 -1.89453125e-01
 -8.30078125e-02  2.05078125e-01 -8.36181641e-03  1.43554688e-01
 -2.85156250e-01  1.22070312e-01 -1.21582031e-01 -2.96875000e-01
 -3.14453125e-01  2.50000000e-01  1.26953125e-01  2.67578125e-01
  3.41796875e-01 -1.34765625e-01 -1.27929688e-01  2.89062500e-01
 -1.28906250e-01 -1.73339844e-02 -1.04003906e-01  2.12402344e-02
  2.27539062e-01 -3.19824219e-02 -1.29882812e-01 -9.66796875e-02
 -2.75390625e-01  6.68945312e-02 -6.49414062e-02 -1.46484375e-01
  9.09423828e-03 -1.81640625e-01 -8.05664062e-02  1.18164062e-01
  1.70898438e-02 -2.57568359e-02 -2.61718750e-01 -4.76562500e-01
  6.93359375e-02 -6.98242188e-02  3.49121094e-02 -2.89916992e-03
  6.00585938e-02  2.13623047e-02  1.99218750e-01 -8.78906250e-02
 -2.53906250e-01 -3.19824219e-02  4.99725342e-04  3.75976562e-02
  1.82617188e-01 -1.61132812e-01 -6.48437500e-01 -7.08007812e-02
  2.11181641e-02 -1.21582031e-01 -2.26562500e-01 -7.81250000e-03
  2.40234375e-01  2.91015625e-01  1.98974609e-02  8.20312500e-02
 -7.27539062e-02  1.11328125e-01  1.14746094e-02 -2.38037109e-03
  3.16406250e-01 -1.80664062e-01 -7.08007812e-02  1.05285645e-03
  1.02233887e-03  2.37304688e-01  4.00390625e-02 -3.10546875e-01
  2.50244141e-02 -1.58203125e-01  1.61132812e-01 -2.64892578e-02
  1.37695312e-01 -2.15820312e-01 -6.93359375e-02  8.20312500e-02
 -1.47460938e-01 -1.02050781e-01 -1.47460938e-01 -1.19140625e-01
 -8.74023438e-02  1.00585938e-01 -2.46582031e-02  1.94335938e-01
 -5.54199219e-02 -4.10156250e-01 -5.37109375e-03 -1.63085938e-01
 -4.06250000e-01  2.06054688e-01 -2.90527344e-02  1.22680664e-02
  4.23828125e-01  1.96289062e-01 -1.70898438e-01 -1.06201172e-02
  2.50000000e-01 -1.87500000e-01  1.31835938e-02  1.76757812e-01
  5.61523438e-02  5.27343750e-02  1.69921875e-01 -1.96289062e-01
  6.16455078e-03  2.47070312e-01 -1.78222656e-02 -1.38671875e-01
  1.45507812e-01  4.02832031e-02  2.57812500e-01  7.66601562e-02
  1.69921875e-01  2.07031250e-01  2.03125000e-01  1.70898438e-01
 -9.52148438e-02 -1.10351562e-01  1.55639648e-02  1.54296875e-01
 -5.95703125e-02  2.69531250e-01 -9.08203125e-02 -8.15429688e-02
  1.16577148e-02 -1.22070312e-01 -2.77343750e-01 -2.16064453e-02
 -2.45117188e-01  4.90722656e-02 -3.20434570e-03  2.09960938e-01
 -1.68945312e-01 -8.66699219e-03 -2.06054688e-01 -1.06445312e-01
  2.39257812e-01  7.61718750e-02 -2.01171875e-01 -7.76367188e-02
 -3.32031250e-01  1.11328125e-01  1.23046875e-01  1.22680664e-02
  4.27734375e-01 -8.83789062e-02 -5.10253906e-02 -2.17285156e-02
 -9.91210938e-02 -2.08984375e-01 -9.52148438e-02 -1.85546875e-01
 -2.27539062e-01 -4.46777344e-02  3.75000000e-01  1.38671875e-01
 -8.44726562e-02  1.90429688e-01  3.08593750e-01 -2.08740234e-02
  2.28515625e-01 -7.03125000e-02 -1.30859375e-01 -2.17285156e-02
 -1.45507812e-01 -5.83496094e-02  5.32226562e-02  3.53515625e-01
  4.00390625e-02 -2.58789062e-02  8.83789062e-02  2.14843750e-02
 -3.80859375e-02  3.37890625e-01  1.55273438e-01 -2.81250000e-01
 -1.65039062e-01  1.97753906e-02  2.08984375e-01 -1.62109375e-01
 -9.91210938e-02  3.50952148e-03 -5.37109375e-02 -1.87500000e-01
 -1.81640625e-01 -7.55310059e-04 -2.61718750e-01 -1.95312500e-01
 -7.12890625e-02 -1.08398438e-01 -3.41796875e-01  2.89062500e-01
  4.16015625e-01  1.92382812e-01  1.55273438e-01 -4.78515625e-02
 -2.98828125e-01  1.16699219e-01 -2.04101562e-01  1.68945312e-01
  6.20117188e-02 -7.51953125e-02  1.35742188e-01  2.46093750e-01
  7.37304688e-02 -1.08032227e-02  2.10937500e-01 -1.44531250e-01
  5.00488281e-02 -9.81445312e-02  1.70898438e-01  2.23632812e-01
 -4.08203125e-01  1.34765625e-01 -3.75000000e-01 -2.34375000e-01
 -1.07910156e-01 -9.61914062e-02 -1.57226562e-01  3.00292969e-02]
```

In [3]:

```
# #编写函数并调用，通过对句子中的词向量求平均的方式获得句子的分布式表示，用于下文分类.
# #获得的训练集的向量矩阵存入train_vecs, 测试集的向量矩阵存入test_vecs.





def wordlist2vecs(corpus):
    res = []
    for wordlist in corpus:
        vecs = np.array([0]*300,dtype='float64')
        count = 0
        for word in wordlist:
            try:
                vecs += model[word]
            except:
                count += 1
        div = np.array([len(wordlist)-count]*300,dtype='float64')
        vecs = np.divide(vecs,div)
        res.append(vecs)

    return np.array(res)


train_vecs = wordlist2vecs(x_train)
test_vecs = wordlist2vecs(x_test)
# try:
#     model.get_index('a')
# except:
#     print('ok')
```

In [4]:

```
#向量缩放
from sklearn.preprocessing import scale
train_vecs=scale(train_vecs)
test_vecs=scale(test_vecs)

#分类
#from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report
from sklearn import  svm
clf = svm.SVC(kernel='linear', gamma=10)
clf.fit(train_vecs,y_train)
y_pred=clf.predict(test_vecs)
print(classification_report(y_test,y_pred))
print('Acc: %.2f' % clf.score(test_vecs, y_test))
```

```
              precision    recall  f1-score   support

           0       0.98      0.94      0.96       609
           1       0.96      0.99      0.97       809

    accuracy                           0.97      1418
   macro avg       0.97      0.96      0.97      1418
weighted avg       0.97      0.97      0.97      1418

Acc: 0.97
```

In [5]:

```

#using SLG to train classifier model, and assess
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import classification_report
lr=SGDClassifier(loss='log',penalty='l1')
lr.fit(train_vecs,y_train)
y_pred = lr.predict(test_vecs)
print(classification_report(y_test,y_pred))

print('Acc: %.2f' % lr.score(test_vecs, y_test))
```

```
c:\Python\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:173: Future
Warning: The loss 'log' was deprecated in v1.1 and will be removed in version 1.3. U
se `loss='log_loss'` which is equivalent.
  warnings.warn(
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       609
           1       0.95      0.94      0.95       809

    accuracy                           0.94      1418
   macro avg       0.94      0.94      0.94      1418
weighted avg       0.94      0.94      0.94      1418

Acc: 0.94
```

第3题 Naive bayes分类和模型评估（30分）

如下垃圾邮件识别代码采用Naive bayes算法实现，请用该模型对data/ SMSSpamCollection.test测试数据进行评测。计算TP,FP, TN, FN的值，以及判断为垃圾邮件的precision, recall, f1的值。

In [1]:

```
# import numpy as np
from collections import Counter
```

In [2]:

```
# 统计训练语料:spam和ham各自邮件总数，单词频率
def seperate(filename):
    hamcnt = Counter()      #ham邮件统计字典
    spamcnt = Counter()     #spam邮件统计字典
    totalNum = 0      # 邮件的总数
    hamNum = 0        # ham邮件数
    spamNum = 0       # spam垃圾邮件数
    i = 0
    file = open(filename, encoding='gb18030', errors='ignore')
    for line in file:  #逐行处理
        i = i + 1
        new = line.split(' ')  # 一行邮件，以ham/spam标记分开
        totalNum = totalNum + 1
        if 'ham' in new[0]:  # ham邮件数
            hamNum = hamNum + 1
            for word in new[1:]:
                hamcnt[word] += 1  #ham中词频计数
        if 'spam' in new[0]:  # spam邮件数
            spamNum = spamNum + 1
            for word in new[1:]:
                spamcnt[word] += 1   #spam词频计数
    print('训练样本的总行数：%s' % i)
    print('ham样本：%s' % hamNum)
    print('spam样本：%s' % spamNum)
    return hamcnt, spamcnt, totalNum, hamNum, spamNum
```

In [3]:

```
train_filename = 'SMSSpamCollection.train'
test_filename = 'SMSSpamCollection.test'

#
hamcnt, spamcnt, totalNum, hamNum, spamNum = seperate(train_filename) #统计spam/ham词典
# 计算spam/ham各自总词数
wordNumerOfham = 0
for key in hamcnt:
    wordNumerOfham += hamcnt[key]
wordNumerOfspam = 0
for key in spamcnt:
    wordNumerOfspam += spamcnt[key]



def predict(test_filename):
    y_test = []
    y_pre = []
    pham,pspam = [],[]
    lines = []
    file = open(test_filename, encoding='gb18030', errors='ignore')
    for line in file:
        # 计算概率p(spam|total),p(ham|total)
        p1_spam = spamNum / totalNum    #ham先验概率
        p1_ham = hamNum / totalNum   #spam先验概率
        hamProbablity = 1
        spamProbability = 1

        newPreData = line.split(' ')
        lines.append(newPreData)

        if 'ham' in newPreData[0]:
            y_test.append('ham')

        if 'spam' in newPreData[0]:
           y_test.append('spam')

        for word in newPreData:   #计算测试语料中每个词的条件概率
            try:  #加1平滑
                hamProbablity = hamProbablity *  (hamcnt[word] + 1) / (wordNumerOfham + len(hamcnt)
            except:  # 文本中没有该单词
                hamProbablity = hamProbablity *  1 / (wordNumerOfham + len(hamcnt))
        res1 = hamProbablity * p1_ham   #为ham类的概率：先验*条件
        pham.append(res1)

        for word in newPreData:
            try:
                spamProbability = spamProbability * (spamcnt[word] + 1) / (wordNumerOfspam + len(sp
            except:
                spamProbability = spamProbability * (1) / (wordNumerOfspam + len(spamcnt))
        res2 = spamProbability * p1_spam  #为spam类的概率：先验*条件
        pspam.append(res2)


        if res1 >= res2:
            y_pre.append('ham')
        else:
            y_pre.append('spam')
    return y_test,y_pre,pham,pspam,lines


y_test, y_pre,pham,pspam,lines = predict(test_filename)
```

```
训练样本的总行数：3345
ham样本：2899
spam样本：446
```

In [4]:

```
def statistics(y_test, y_pre):
    tp,fp,tn,fn = 0,0,0,0

    if len(y_test) == len(y_pre):
        # print('In')
        for i in range(len(y_pre)):
            # tp
            if y_pre[i] == 'spam' and y_test[i] == 'spam':
                tp += 1
            # fp
            elif y_pre[i] == 'spam' and y_test[i] == 'ham':
                fp += 1
            # tn
            elif y_pre[i] == 'ham' and y_test[i] == 'ham':
                tn += 1
            # fn
            elif y_pre[i] == 'ham' and y_test[i] == 'spam':
                fn += 1
            else:
                print("error")


    else:
        raise ValueError("two lists have different lengths!")

    return tp,fp,tn,fn


tp,fp,tn,fn = statistics(y_test, y_pre)
precision = tp/(tp+fp)
recall = tp/(tp+fn)
f1 = (2*precision*recall) / (precision + recall)
acc = (tp + tn) / (tp+tn+fp+fn)

print('tp,fp,tn,fn:',tp,fp,tn,fn)
print('precision:',precision)
print('recall:',recall)
print('f1:',f1)
print('acc:',acc)
```

```
tp,fp,tn,fn: 141 193 776 4
precision: 0.4221556886227545
recall: 0.9724137931034482
f1: 0.5887265135699373
acc: 0.8231597845601436
```

第4题 思考题。字数不限(10分) ¶

基于NLP的发展和技术革新，就某一具体案例提出自己的思考，案例可以体现国家情怀、社会责任、人文精神、诚信品质、创新能力的等积极品质的内容。

Modeling Dual Read/Write Paths for Simultaneous Machine Translation

探讨机器同声传译中自然语言处理问题和解决方法，以及其中带来的新的启发

论文试图解决什么问题?

同声传译需要在接受源句子的同时输出翻译，因此模型需要指定一个同传策略来决定是读入下一个源词还是输出目标词。然而，两种语言之间的同传策略复杂且难以学习。本文试图让模型从自身内部挖掘同传策略的监督信号，学习到更好的同传策略。

这篇文章要验证一个什么科学假设?

本文希望能够验证：同声传译模型可以从自身内部挖掘同传策略的监督信号，以自学习的方式学习到更准确的同传策略。

论文中提到的解决方案之关键是什么?

学习更好的同声传译策略的关键是挖掘模型内部的约束来为同传策略提供明确的监督信号。具体地，本文首先指出两个翻译方向的源端读操作应满足充分必要性，即连续读入的源端片段和随后生成的目标片段应该互为翻译，此为两个翻译方向的对偶特性，然后，本文提出Dual-Path SiMT来联合训练两个翻译方向上的同声传译模型，同时约束两个方向的同传策略之间的对偶约束。

这篇论文到底有什么贡献?

本文提出了一种同传策略的自监督学习方法，借助于双向翻译的对偶特性，第一次实现了无需外部信息即可从模型自身挖掘明确的读/写监督信号。

下一步呢？有什么工作可以继续深入?

下一步，同声传译的研究可以继续深入探索如何开发更精确的同传策略，从而获得更好的同声传译性能。另外，未来工作还可以着手于增强同声传译模型在源信息受限的情况下的翻译能力。