基于卷积神经网络的文本情感分类

> 注：上述题目是网上申请时提交的题目。后来加以引申，最终项目准确的名称应该是 “实现基于卷积神经网络和支持向量机的文本情感分类，并讨论数据量对这两种方式准确率的影响。”

内容

1、通过网络爬虫从淘宝、京东爬取商品评论，对其进行人工分类、标注，得到数据集。

2、对数据集进行预处理，进行中文分词、去除停用词，标点符号。

3、分别使用 “卷积神经网络” 和 “支持向量机” 进行训练和预测，计算准确率。

4、改变用于训练的数据量大小，比较这两种方式下的准确率变化情况。

依赖库

```
python=3.6
pandas=0.22.0
numpy=1.14.0
jieba=0.39
gensim=3.4.0
scikit-learn=0.19.1
keras=2.1.5
wordcloud=1.4.1
```

具体实现

构建数据集

1、 原始数据爬取

 创建商品评论页面的循环链接，使用Python的requests库循环抓取数据，使用正则表达式匹配查询，得到相关文本，将结果保存为.csv文档。

```
for url in urls:
    content = requests.get(url).text
# 借助正则表达式使用findall进行匹配查询
    nickname.extend(re.findall('"displayUserNick":"(.*?)"',content)) #用户名
    ratecontent.extend(re.findall(re.compile('"rateContent":"(.*?)","rateDate"'),content)) #评论内容
    ratedate.extend(re.findall(re.compile('"rateDate":"(.*?)","reply"'),content)) #评论时间
```

2、标注情感

正面:

```
使用了一个月了，大品牌就是好，满意的没话说，给全五分好评。,正面
挺好 就配送太忙了 已经到了好几天 非要在约定日子配送 提前还不能提前,正面
好好好好好,正面
电视不错 带蓝牙语音功能。如果有需要。下次还会选择暴风。,正面
不错,正面
电视效果很好,正面
感觉很好，性价比很高。 晒单： 收起 向左转 向右转,正面
秒杀1899，以前没有用过不知好不好用。,正面
海信电视，真的很好！性价比很高！推荐大家使用。
客服好，商家好，送货的也好，上门安装的师傅更好，我是55寸座机，安装调试仅收人民币30元，合规合理。明天再买一个。,正面
不错，物美价廉，能再清晰一点就好了！,正面
电视机不错，清晰度也好，有WiFi高能，直接网络播放，我是满意的，还有会员送，坐等会员。,正面
质量不错，效果好，用一段时间再评论。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。,正面
京东物流送货快，双十二活动上午抢到的，价钱比平时优惠很多，安装师傅服务态度好，挂墙配挂架没有乱收费现象。这款电视款式
买,正面
质量非常不错，大气又实惠。服务可以，收费合理,正面
电视不错哦，就是手机太差拍的不好，非常的喜欢，看电视流畅，用了几天才来评价的,正面
电视非常的好，很喜欢。性价比超高，从快递到安装服务都很满意。全部给满分。,正面
```

负面:

```
好大呀看一会眼睛就有点不舒服，声音没感觉震撼，清晰度可以。我还没开通会员。,负面
画面不错，这款就是不带dts和杜比音效，外放很一般，自己配上音响效果好不少,负面
画面不错非常清晰，图案也细腻，主要音效很好像安上音响一般的效果，开机时间长一点,负面
画面挺清晰的，该有的功能都有了，而且是高配，就是声音有点失望，相比5年前买的sony差不是一点.支持国产，仍需努力.,负面
画质还行，没有漏光现象。就是侧面看会有泛白，颜色失真。音质也没有宣传的那么好。电视主页面跟电视猫页面很相似。跟电视猫界面来回切换的
，很容易搞混。电视面板真的很薄，很怕屏幕会变形。蓝牙遥控还是蛮好用的。总体感觉还满意。就是不知能用多少年？,负面
家里人都说好，很清晰，看高清电视很爽，音响一般，看电视够了，有追求的还是再上别的音响。,负面
家里微鲸已经第三台了，16年买的55k1没想到现在涨到四千多了，姐家买的65的，自己新家一直想买个微鲸的奈何涨价太贵。终于等到618发现个
宜的，一问知道是京东方的屏幕，放心购买，因为家里上一个32的电视就是京东方的，质量不错！昨天拍了这个今天就到了。除了音箱没有我家那
好以外其他都不错！,负面
```

分词

 在英文的行文中，单词之间是以空格作为自然分界符的，而中文只是字、句和段能通过明显的分界符来简单划界，但是词却没有一个形式上的分界符。要获得句子的语义特征，必须对句子进行分词。

使用"jieba分词"实现: `words=jieba.lcut(text)`

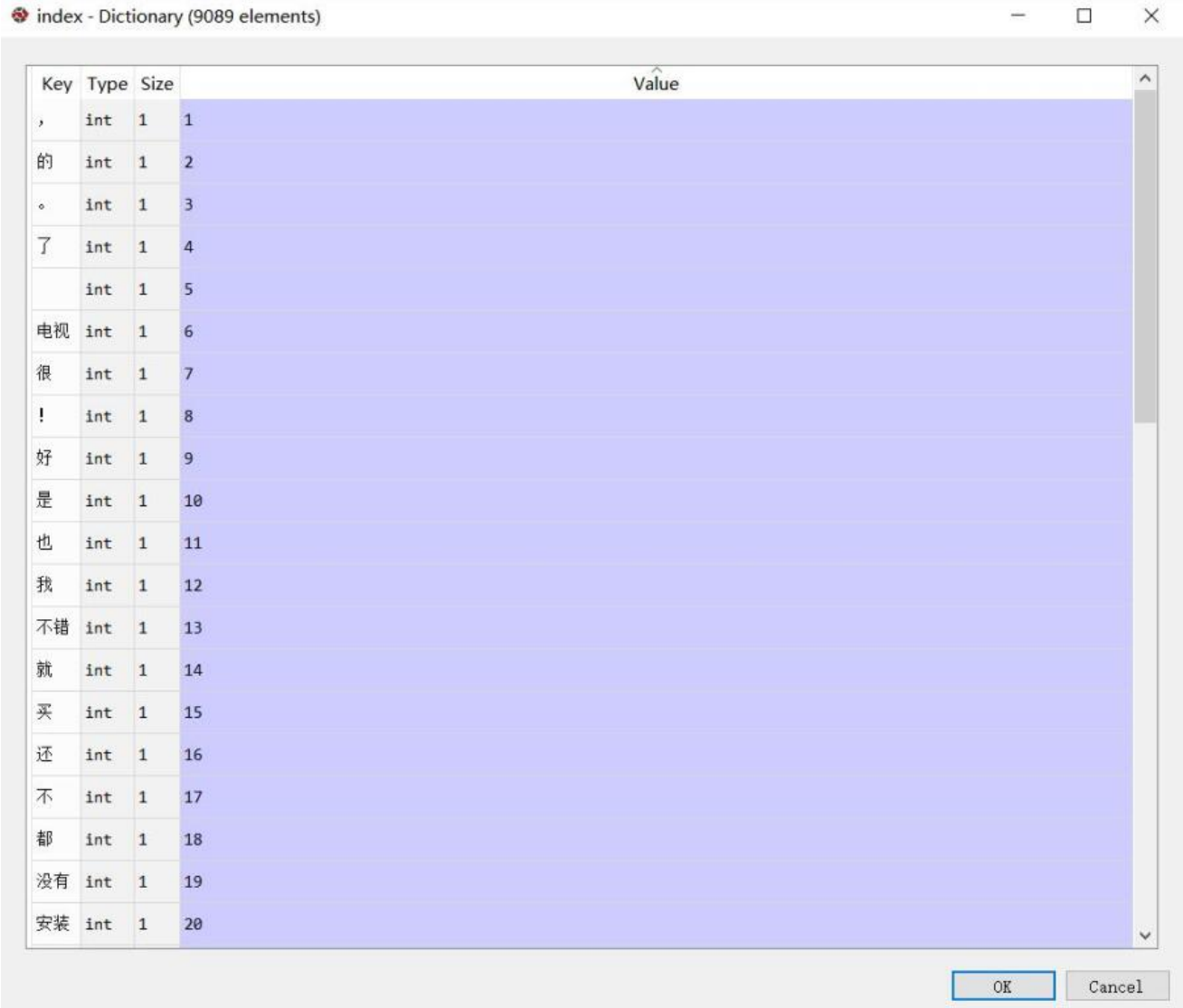
分词结果

使用基于wordcloud生成的词云表示分词结果:

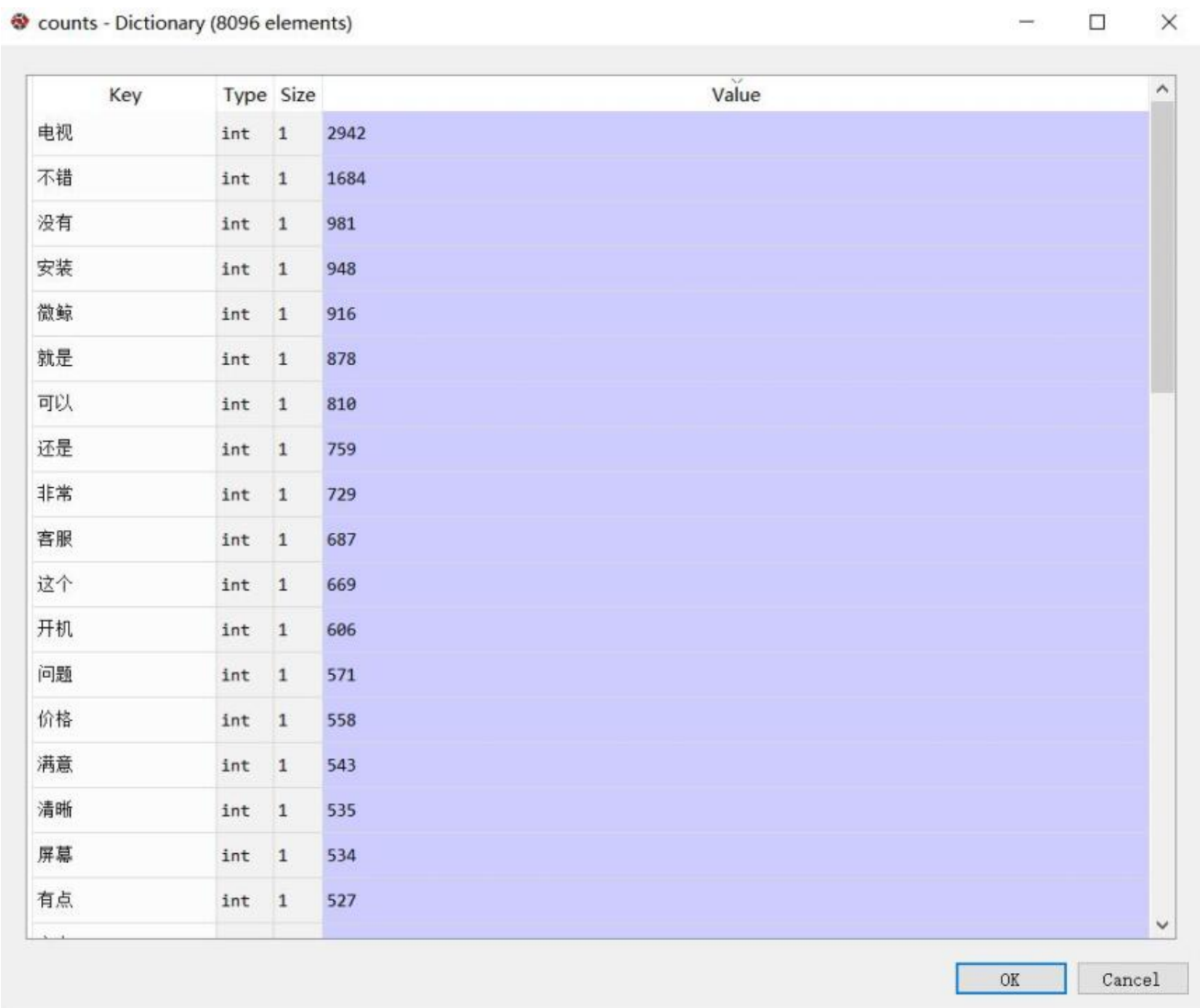
```
wc= WordCloud(font_path=font_path,
    background_color='white',
    max_words=2000,
    mask=back_coloring,
    max_font_size=100,
    random_state=30,
    width=600,height=400,margin=5,
    relative_scaling=0.5
    )
wc.generate(text)
```

关于最小词长的讨论

最小词长为1时的词频：



最小词长为2时的词频：



由图可见，当最小词长选为1时，高频词出现了大量的标点符号、语气词等无关词语。虽然可以通过通用词将其剔除，但是这么做比较复杂，并且需要构建一个庞大的停用词表。选最小词长为2时，在高频词中明显可以看到很多表示情感的词，比如像"不错"、"满意"等等，已经满足了需求。所以最终选择的最小词长为2。

卷积神经网络

代码见 CNN.py

流程： 导入数据 -> 数据预处理（分词，统计词频、编号，标签转成独热码） -> 构建模型 -> 训练和预测 -> 计算准确率

数据预处理部分

1、分词（同上）

2、统计词频、编码

使用Keras的Tokenizer类 对文本中的词进行统计计数，生成文档词典，以支持基于词典位序生成文本的向量表示。

```
tokenizer = Tokenizer(num_words=2000) #初始化Tokenizer
tokenizer.fit_on_texts(texts=X_cut) #训练
counts=tokenizer.word_counts #词频
index=tokenizer.word_index #编号
```

> 注：在Keras官方文档中，Tokenizer被翻译成“分词器”，这个翻译其实不太恰当。它实现的功能并不是分词，而是统计词频，并且可以根据词频从高到低对词语依次编码为1,2,3……。

3、标签转成独热码

使用Pandas的get_dummies方法实现。

```
y_one_hot = pd.get_dummies(y)
y_one_hot_labels = np.asarray(y_one_hot)
```

模型构建部分

模型来自博文（Implementing a CNN for Text Classification in TensorFlow：http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/ ）。因为要做的只是二分类，所以对其模型中平行的三个卷积层进行了简化，只使用一个卷积层。 因为发现过拟合严重，增加了两个Dropout层。

```
data_input = Input(shape=[maxlen])
word_vec = Embedding(input_dim=num_words+1,
                     input_length=maxlen,
                     output_dim=vec_size,
                     mask_zero=0)(data_input)
x = Conv1D(filters=128, kernel_size=[3], strides=1, padding='same', activation='relu')
(word_vec)
x = GlobalMaxPool1D()(x)
x = Dropout(0.1)(x)
x = Dense(500, activation='relu')(x)
x = Dropout(0.1)(x)
x = Dense(output_shape, activation='softmax')(x)
model = Model(inputs=data_input, outputs=x)
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

训练和预测部分

```
#训练
model.fit(x=X_train, y=y_train, epochs=3, batch_size=64,verbose=2)
#预测
y_predict = model.predict(X_test)
#转换预测结果
y_predict_label = label2tag(predictions=y_predict, y=y)
#统计正确率
Y_test=label2tag(predictions=y_test,y=y)
print(sum([y_predict_label[i] == Y_test[i] for i in range(len(y_predict))]) / len(y_predict))
```

支持向量机

代码见 SVM.py

流程： 导入数据 -> 数据预处理（分词，词语转为词向量） -> 构建模型 -> 训练和预测 -> 计算准确率

数据预处理部分

1、分词，同上

2、词向量生成

使用中文词向量语料库（Shen Li, Zhe Zhao, Renfen Hu, Wensi Li, Tao Liu, Xiaoyong Du, *Analogical Reasoning on Chinese Morphological and Semantic Relations*, ACL 2018.）中的中文预训练词向量（sgns.weibo.word）对分词结果进行转换。

```
model = gensim.models.KeyedVectors.load_word2vec_format(word2vec_loadpath,binary=False)
text_vec = [[model[word] for word in text_cut if word in model] for text_cut in X_cut]
```

模型构建部分

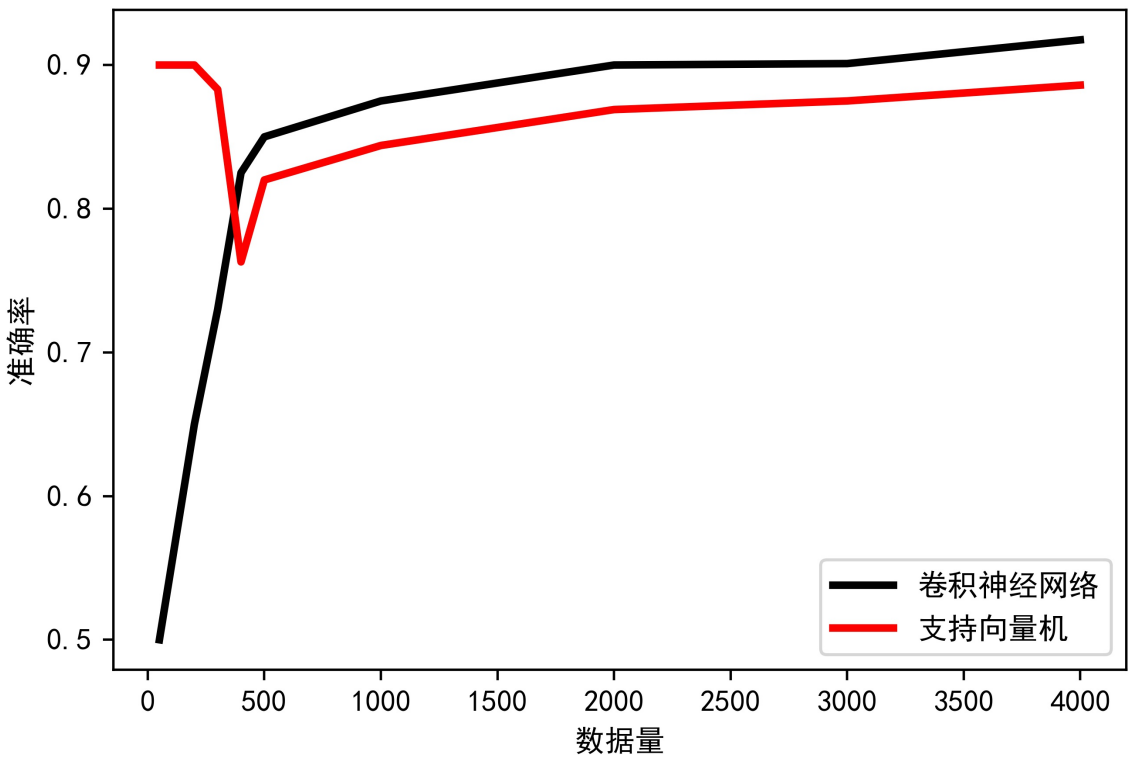
```
model_svc = SVC(C=10,kernel='linear',)
#C似乎越大准确率越高： score 0.819672 (C=1)  0.83957(C=2)  0.850117(C=3)  0.862997(C=5)
#尝试了所有的kernel，最后得出linear的准确率最高。
```

训练和预测部分

```
model_svc.fit(X=X_train,y=y_train,)
y_predict  = model_svc.predict(X_test)
print(sum(y_predict == np.array(y_test)) / len(y_predict))
```

准确率比较

数据量	卷积神经网络	支持向量机
50	0.500	0.900
100	0.550	0.900
200	0.650	0.900
300	0.733	0.883
400	0.825	0.763
500	0.850	0.820
1000	0.875	0.844
2000	0.900	0.869
3000	0.901	0.875
4000	0.9175	0.886



可以看出，卷积神经网络随着数据量的增加，正确率也是不断地提高，最后收敛在0.90左右。而支持向量机的正确率却是经历了先减小后增大的过程。 说明对于卷积神经网络，用于训练的数据量越大，效果越好。而对于支持向量机，更重要的可能是参数的设置。

附录

数据爬取部分代码：

```
# 导入所需的开发模块
import requests
import re
# 创建循环链接
urls = []
for i in list(range(1,100)):
    urls.append('https://rate.tmall.com/list_detail_rate.htm?itemId=560135062971&spuId=889297741&sellerId=1714128138&order=3&currentPage=%s' %i)

# 构建字段容器
nickname = []
ratedate = []
ratecontent = []
i=0
# 循环抓取数据
for url in urls:
    content = requests.get(url).text

# 借助正则表达式使用findall进行匹配查询
    nickname.extend(re.findall('"displayUserNick":"(.*?)"',content))
#    color.extend(re.findall(re.compile('颜色分类:(.*?);'),content))
    ratecontent.extend(re.findall(re.compile('"rateContent":"(.*?)","rateDate"'),content))
    ratedate.extend(re.findall(re.compile('"rateDate":"(.*?)","reply"'),content))
    print(i)
    i=i+1
# 写入数据

file = open('xiaomi5A_comment.csv','w')
for i in list(range(0,len(nickname))):
    file.write(','.join((nickname[i],ratedate[i],ratecontent[i]))+'\n')
file.close()

```

卷积神经网络部分代码：

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Model
from keras.layers import Dense, Embedding, Input
from keras.layers import Conv1D, GlobalMaxPool1D, Dropout
import jieba

from pandas.core.frame import DataFrame
```

```
11
12  def cut_texts(texts=None, word_len=1, savename=None):
13      #分词
14      texts_cut=[]
15      text_one=[]
16      if word_len > 1:
17          for text in texts:
18              text_cut=[]
19              words=jieba.lcut(text)
20              for word in words:
21                  if len(word)>=word_len:
22                      text_cut.append(word)
23                      text_one.append(word)
24              texts_cut.append(text_cut)
25      else:
26          for text in texts:
27              words=jieba.lcut(text)
28              for word in words:
29                  text_one.append(word)
30              texts_cut.append(words)
31      if savename is not None:
32          file=open(savename,'w',encoding='utf-8')
33          file.write(' '.join(text_one))
34          file.close()
35      return texts_cut
36
37  def text2seq(texts_cut=None, maxlen=30,tokenizer=None):
38      #文本转序列
39      fact_seq = tokenizer.texts_to_sequences(texts=texts_cut)
40      print('finish texts to sequences')
41      fact_pad_seq = []
42      # pad_sequences,将每一条评论都填充（pad）到一个矩阵中。 最大长度30，超出长度从前面截断。结尾补
    0。
43      fact_pad_seq += list(pad_sequences(fact_seq, maxlen=maxlen,
44          padding='post', truncating='pre', value=0, dtype='int'))
45      return fact_pad_seq
46
47  def label2tag( predictions,y):
48      label_set = []
49      for i in y:
50          label_set.append(i)
51      label_set=np.array(list(set(label_set)))
52
53      labels = []
54      for prediction in predictions:
55          label = label_set[prediction == prediction.max()]
56          labels.append(label.tolist())
57
58      return labels
59
60  #导入数据
61  data=pd.read_csv("1000.csv")

62  x = data['evaluation']
```

```
y=data['label']
#分词
X_cut= cut_texts(texts=x, word_len=2,savename='ciyun.txt')

# 对文本中的词进行统计计数，生成文档词典，以支持基于词典位序生成文本的向量表示。
tokenizer = Tokenizer(num_words=2000)
tokenizer.fit_on_texts(texts=X_cut)
#index=tokenizer.word_index
#counts=tokenizer.word_counts

#文本转矩阵
maxlen=20 #矩阵维度
X_seq = text2seq(texts_cut=X_cut,maxlen=maxlen, tokenizer=tokenizer)
X_seq = np.array(X_seq)

#标签转成独热码
y_one_hot = pd.get_dummies(y)
y_one_hot_labels = np.asarray(y_one_hot)

#分割训练集、测试集
X_train, X_test, y_train, y_test = train_test_split(X_seq, y_one_hot_labels, test_size=0.2)

num_words = 2000
vec_size = 128
output_shape = 2

#构建模型
data_input = Input(shape=[maxlen])
word_vec = Embedding(input_dim=num_words+1,
                     input_length=maxlen,
                     output_dim=vec_size,
                     mask_zero=0)(data_input)
x = Conv1D(filters=128, kernel_size=[3], strides=1, padding='same', activation='relu')
(word_vec)
x = GlobalMaxPool1D()(x)
x = Dropout(0.1)(x)
x = Dense(500, activation='relu')(x)
x = Dropout(0.1)(x)
x = Dense(output_shape, activation='softmax')(x)
model = Model(inputs=data_input, outputs=x)
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()

#训练
model.fit(x=X_train, y=y_train, epochs=3, batch_size=64,verbose=2)
#model.fit(x=X_train, y=y_train, epochs=3, batch_size=64,verbose=2,validation_split=0.1)
#预测
y_predict = model.predict(X_test)
#转换预测结果
y_predict_label = label2tag(predictions=y_predict, y=y)

#统计正确率
```

```
Y_test=label2tag(predictions=y_test,y=y)
print(sum([y_predict_label[i] == Y_test[i] for i in range(len(y_predict))]) /
len(y_predict))

```

支持向量机部分代码

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import jieba
from sklearn.svm import SVC
import gensim

def cut_texts(texts=None, word_len=1, savename=None):
    #分词
    texts_cut=[]
    text_one=[]
    if word_len > 1:
        for text in texts:
            text_cut=[]
            words=jieba.lcut(text)
            for word in words:
                if len(word)>=word_len:
                    text_cut.append(word)
                    text_one.append(word)
            texts_cut.append(text_cut)
    else:
        for text in texts:
            words=jieba.lcut(text)
            for word in words:
                text_one.append(word)
            texts_cut.append(words)
    if savename is not None:
        file=open(savename,'w',encoding='utf-8')
        file.write(' '.join(text_one))
        file.close()
    return texts_cut

#导入数据
data=pd.read_csv("300.csv")
x = data['evaluation']
y = data['label']

#分词并解决分词后空白list的问题
X_cut_n= cut_texts(texts=x, word_len=2)
X_cut=[]
label=[]
for i in range(0,len(X_cut_n)):
    if (len(X_cut_n[i])!=0) :
        X_cut.append(X_cut_n[i])

        label.append(y[i])
```

```
del X_cut_n
del y

# 转为词向量，去掉空list、
word2vec_loadpath='sgns.weibo.word'
needSave='False'
model = gensim.models.KeyedVectors.load_word2vec_format(word2vec_loadpath,binary=False)
if needSave:
    model.save('word2vec_model')
text_vec = [[model[word] for word in text_cut if word in model] for text_cut in X_cut]
text=[]
labels=[]
for index,i in enumerate(text_vec):
   if len(i)!=0:
        text.append(sum(i)/len(i))
        labels.append(label[index])
text=np.array(text)
# texts vector
#x_vec = np.array([sum(i) / len(i) for i in x_word_vec])
X_train, X_test, y_train, y_test = train_test_split(text, labels,
test_size=0.2,random_state=1)

#构造SVC模型
model_svc = SVC(C=10,kernel='linear',)
#训练和预测
model_svc.fit(X=X_train,y=y_train,)
y_predict  = model_svc.predict(X_test)
print(sum(y_predict == np.array(y_test)) / len(y_predict))

```

词云生成代码：

```
from os import path
from scipy.misc import imread
import matplotlib.pyplot as plt
import jieba
from wordcloud import WordCloud,ImageColorGenerator

def jiebaclearText(text):
    #分词
    myword_list=[]
    seg_list = jieba.cut(text, cut_all=False)
    liststr='/'.join(seg_list)
    for myword in liststr.split('/'):
        if len(myword.strip())>1 :
            myword_list.append(myword)
    return ' '.join(myword_list)

d=path.dirname('.') #获取当前文件路径
stopwords={} #停用词表
isCN=0 # 是否需要分词

back_coloring_path = "img.jpg" #设置背景图片路径
```

```
text_path="ciyun_single.txt" #设置要分析的文本路径
font_path= 'STXINWEI.TTF' #中文字体路径
imgname1 = 'WordCloudDefautColors.png' #保存图片1 ， 只按照背景图片形状
imgname2= 'WordCloudColorByImg.png' #保存图片名字2 ， 颜色按照背景图片颜色布局生成
back_coloring=imread(path.join(d,back_coloring_path)) #设置背景图片

#设置词云属性
wc= WordCloud(font_path=font_path,
    background_color='white',
    max_words=2000,
    mask=back_coloring,
    max_font_size=100,
    random_state=30,
    width=600,height=400,margin=5,
    relative_scaling=0.5
    )

text=open(path.join(d,text_path),encoding='utf-8').read()

if isCN:
    text=jiebaclearText(text)

#生成词云
wc.generate(text)

#从背景图片生成颜色值
image_colors=ImageColorGenerator(back_coloring)
#绘图
plt.figure()
plt.imshow(wc)
plt.axis('off')
plt.show()
wc.to_file(path.join(d,imgname1))
img_colors=ImageColorGenerator(back_coloring)
plt.imshow(wc.recolor(color_func= image_colors))
plt.axis('off')
plt.figure()
plt.imshow(back_coloring,cmap=plt.cm.gray)
plt.axis('off')
plt.show()
wc.to_file(path.join(d,imgname2))
```