情感分析是一种常见的自然语言处理（NLP）方法的应用，特别是在以提取文本的情感内容为目标的分类方法中。通过这种方式，情感分析可以被视为利用一些情感得分指标来量化定性数据的方法。尽管情绪在很大程度上是主观的，但是情感量化分析已经有很多有用的实践，比如企业分析消费者对产品的反馈信息，或者检测在线评论中的差评信息。

数据集来源：
https://github.com/BUPTLdy/Sentiment-Analysis

数据集文件：

好评数据：pos.xls
差评数据：neg.xls

好评数据样例：

	A
10650	简单，安装很快，电话打了半小时就来装了
10651	挺好，卖家态度也很好
10652	东西还不错　　自己装起来了　　希望能用久点
10653	很好用，快递很给力，帮我送到目的地
10654	用过了才来评价，挺好用的和我在商场买的一样，应该是正品，烧水也快。五分。
10655	17号付款，19号中午到货，下午售后就给安上了。今天早上就洗澡用了。客服大海、送货的中通快递以及售后安装人员服务都极好。材料费93，不贵。很满意的一次购物，
10656	第二次购买了，店家很热情，宝贝很完美，烧水挺快的，好评，嘻嘻！　服务很热情，点100个赞，哈哈！！！
10657	买来放在出租房里的，所以自己也没试过，但是安装服务人员特别好，最大限度地给省钱，两套热水澡装下来，一共只收了150块的材料费，还帮着调试好，再三检查后才离开，点个赞，相信美的。
10658	买来放在出租房里的，所以自己也没试过，但是安装服务人员特别好，最大限度地给省钱，两套热水澡装下来，一共只收了150块的材料费，还帮着调试好，再三检查后才离开，点个赞，相信美的。
10659	东西不错，装上去用了
10660	东西收到了很满意，快递也很快。。。。
10661	超级好的卖家！之前不小心拍错了，客服非常耐心的帮我解答问题，快递也非常给力，必须赞！！！
10662	还没拆货，物流给力，送货到家，一直信赖大品牌，给32个赞。
10663	挺好的，卖家态度也很好
10664	挺好的，今天用了。
10665	很好的热水器！很实际的功能！大品牌！　很好！安装花了90元！很好！
10666	好评！　好评！
10667	质量不错~~~~
10668	东西很好 妈妈很喜欢 大天客服 服务很好 等安装后在来追加评论
10669	发货很快，物流也及时，热水器包装很好，已经打电话找师傅安装好了，用了加热挺快的，非常好的热水器
10670	很不错，安装很到位哦！
10671	东东不重，一个人就拉回家了，找人来安装，费用也不高，热水器很精致，哈哈，物有所值，很满意，当晚就使用上了。 好评，全5分。
10672	安装好了 挺好的！赞
10673	宝贝很好很喜欢　卖家有多努力哦~~~~ 好

差评数据样例：

	A
1	做为一本声名在外的流行书，说的还是广州的外企，按道理应该和我的生存环境差不多啊。但是一看之下，才发现相去甚远。这也就算了，还发现其中的很多规则有很强的企业个性，也就说，
2	作者有明显的自恋倾向，只有有老公养不上班的太太们才能像她那样生活。很多方法都不实用，还有抄袭嫌疑，没意思，没买的可以省省了。
3	作者完全是以一个过来的自认为是成功者的角度去写这个问题，感觉很不客观。虽然不是很喜欢，但是，有不少的观点还是可取的。
4	作者提倡内调，不信任化妆品，这点赞同。但是所列举的方法太麻烦，配料也不好找。不是太实用。
5	作者的文笔一般，观点也是和市面上的同类书大同小异，不推荐读者购买。
6	作者的文笔还行，但通篇感觉太琐碎，有点文人的无病呻吟。自由主义者。作者的品性不敢苟同，无民族立场（可谓之汉奸），风流成性
7	作者倒是个很小资的人,但有点自恋的感觉,书并没有什么大帮助
8	作为一本描写过去年代感情生活的小说，作者明显生活经验不足，并且文字功底极其一般，看后感觉浪费时间，不值得看，也不推荐。张贤亮，梁晓声等写作大家叙述的过去时光似乎更让人怀
9	作为个人经验在网上谈谈可以，但拿来出书就有点过了，书中还有些明显的谬误。不过文笔还不错，建议写写散文好了。
10	昨天刚兴奋地写了评论,今天便遇一闹心事,因把此套书推荐给很多朋友,朋友就拖我在网上购,结果前天购了三套,送货倒挺快,可只给了三本1,没有2,为什么一张订单上的货不全还发呢?已发了
11	昨天打开书才翻了几页，书页纷纷掉落了，请问怎么回事？
12	最近下单号为：1442083355，其中一套书《易经的智慧》当当网　少带光盘　。为了得到解决，我去信两次，二三天无人回复。　然后我打电话[08/1/04 1:30]010-51236699客服热线，分机
13	最大的缺陷是容易诱导养成中文式英语习惯
14	最大的困扰是穴位有点把不准，不过我牢记中里先生离穴不离经的教诲。
15	最大的不好就是没有音标，看的时候不是很方便，而且书的内容一点也不好笑，或者说作者幽默感很有限。。。
16	纵观整部书（上下两册）　从文字，到结构，人物，情节 没有一个地方是可取的虽然有过从业经验 但并不是有过类似的经历就可以写书的文字的东西　需要表达　需要雕琢很失望的一本书
17	总结一下，女人要现实。要么嫁好老公，要么抓住机遇有事业……这还用她说啊。
18	总共买了8本书，其它都还好，这本书感觉旧旧的，也没什么心情去看了
19	总的来说有点乱七八糟的感觉。重复又重复。

环境参数：

```
gensim ==3.8.0
sklearn ==0.20.2
Keras ==2.1.2
```

一.词向量的生成

为了生成词向量模型，需要借助gensim框架，gensim框架的基础应用：
Demo1：

```
from gensim.models import word2vec
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
#准备数据集
raw_sentences = ['the quick brown fox jumps over the lazy dogs',
'yoyoyo cat go home now to sleep']
#分词
sentences = [s.split() for s in raw_sentences]
#构建模型
model = word2vec.Word2Vec(sentences,min_count = 1)
#相似度比较
print(model.similarity('dogs','cat'))
```

运行结果：-0.078486055

二，LSTM算法的训练与预测演示

实现步骤：

1.加载数据集，然后分词。
2.统计词频，生成词向量。
3.对每条评论数据进行长度的padding，生成词向量映射。
4.基于sklearn框架划分训练集(80%)和测试集(20%)，开始训练LSTM算法模型。
5.基于模型预测新的数据集。

1.加载算法框架

```
import jieba
import sys
import yaml
import multiprocessing
import numpy as np
import pandas as pd
from sklearn.model_selection   import train_test_split
from gensim.models.word2vec    import Word2Vec
from gensim.corpora.dictionary import Dictionary

from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers.embeddings import Embedding
from keras.layers.recurrent  import LSTM
from keras.layers.core       import Dense,Dropout,Activation
from keras.models            import model_from_yaml
```

2.设置算法参数:

```
np.random.seed(666)
#设置参数
vocab_dim    = 100
maxlen       = 100
n_iterations = 1
min_frequence  = 10
window_size  = 5
batch_size   = 32
n_epoch      = 1
input_length = 100
cpu_count    = multiprocessing.cpu_count()
```

3.数据集准备与处理

```
#加载数据文件
def loadfile():
        neg = pd.read_excel('data/neg.xls',header = None,index = None)
        pos = pd.read_excel('data/pos.xls',header = None,index = None)
        x = np.concatenate((pos[0],neg[0]))
        #1表示好评，0表示差评
        y = np.concatenate((np.ones(len(pos),dtype = int),np.zeros(len(neg),dtype = int )))
        return x,y

#对句子进行分词，并去掉换行符
def tokenizer(text):
        text = [jieba.lcut(sentence.replace('\n','')) for sentence in text]
        return text

#创建词语字典，并返回每个词语的索引，词向量，以及每个句子所对应的词语索引
def create_dict(model = None,x = None):
        #1- Creates a word to index mapping
        #2- Creates a word to vector mapping
        #3- Transforms the Training and Testing Dictionaries
        if (x is not None) and (model is not None):
                gensim_dict = Dictionary()
                gensim_dict.doc2bow(model.wv.vocab.keys(),allow_update = True)
        #所有频数超过10的词语的索引
                word2index  = {v:k+1 for k,v in gensim_dict.items()}
        #所有频数超过10的词语的词向量
                word2v = {word:model[word] for word in word2index.keys()}

                def parse_dataset(x):
                        #Words become integers
                        data = []
                        for sentence in x:
                                new_sentence = []
                                for word in sentence:
                                        try:
                                                #将句子中的每个单词转换成对应的索引
                                                new_sentence.append(word2index[word])
                                        except:
                                                #句子中含有频数小于10的词语，索引为0
                                                new_sentence.append(0)
                                data.append(new_sentence)
                        return data
                x = parse_dataset(x)
                x = sequence.pad_sequences(x,maxlen = maxlen)
                return word2index,word2v,x
```

```
        else:
                print ('No data provided...')

#训练词向量模型，并返回每个词语的索引，词向量，以及每个句子所对应的词语索引
def word2vec_train(x):
        model = Word2Vec(
                        size      = vocab_dim,
                        min_count = min_frequence,
                        window    = window_size,
                        workers   = cpu_count,
                        iter      = n_iterations
                )
        model.build_vocab(x)
        model.train(x,total_examples = model.corpus_count,epochs = model.iter)
        model.save('lstm_data/Word2vec_model.pkl')
        index_dict, word_vectors,x = create_dict(model=model,x=x)
        return index_dict, word_vectors,x

def get_data(index_dict,word_vectors,x,y):
    # 所有单词的索引数，频数小于10的词语索引为0，所以加1
    n_symbols = len(index_dict) + 1
    #索引为0的词语，词向量全为0
    embedding_weights = np.zeros((n_symbols, vocab_dim))
    #从索引为1的词语开始，对每个词语对应其词向量
    for word, index in index_dict.items():
        embedding_weights[index, :] = word_vectors[word]
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
    print (x_train.shape,y_train.shape)
    return n_symbols,embedding_weights,x_train,y_train,x_test,y_test
```

4.模型训练与预测

```
##定义网络结构
def train_lstm(n_symbols,embedding_weights,x_train,y_train,x_test,y_test):
    print ('Defining a Simple Keras Model...')
    model = Sequential()  # or Graph or whatever
    model.add(Embedding(output_dim=vocab_dim,
                        input_dim=n_symbols,
                        mask_zero=True,
                        weights=[embedding_weights],
                        input_length=input_length))  # Adding Input Length
    model.add(LSTM(output_dim=50, activation='sigmoid', inner_activation='hard_sigmoid'))
    model.add(Dropout(0.5))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))

    print ('Compiling the Model...')
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',metrics=['accuracy'])

    print ("Train...")
    model.fit(x_train, y_train,
            batch_size=batch_size,
            nb_epoch=n_epoch,verbose=1,
            validation_data=(x_test, y_test))

    print ("Evaluate...")
    score = model.evaluate(x_test, y_test,batch_size=batch_size)

    yaml_string = model.to_yaml()
    with open('lstm_data/lstm.yml', 'w') as outfile:
        outfile.write( yaml.dump(yaml_string, default_flow_style=True) )
    model.save_weights('lstm_data/lstm.h5')
    print ('Test score:', score)

#训练模型，并保存
def train():
    print ('Loading Data...')
    x,y=loadfile()
    print (len(x),len(y))
    print ('Tokenising...')
    x = tokenizer(x)
    print ('Training a Word2vec model...')
    index_dict, word_vectors,x=word2vec_train(x)
    print ('Setting up Arrays for Keras Embedding Layer...')
    n_symbols,embedding_weights,x_train,y_train,x_test,y_test=get_data(index_dict,
```

```
    word_vectors,x,y)
    print (x_train.shape,y_train.shape)
    train_lstm(n_symbols,embedding_weights,x_train,y_train,x_test,y_test)

def input_transform(string):
    words=jieba.lcut(string)
    words=np.array(words).reshape(1,-1)
    model=Word2Vec.load('lstm_data/Word2vec_model.pkl')
    _,_,x=create_dict(model,words)
    return x

def lstm_predict(string):
    with open('lstm_data/lstm.yml', 'r') as f:
        yaml_string = yaml.load(f)
    model = model_from_yaml(yaml_string)
    model.load_weights('lstm_data/lstm.h5')
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',metrics=['accuracy'])
    data=input_transform(string)
    data.reshape(1,-1)
    result=model.predict_classes(data)
    if result[0][0]==1:
        print (string,'\n AI分析结果：  好评')
    else:
        print (string,'\n AI分析结果：  差评')
if __name__=='__main__':
        train()
        string ='牛逼的手机，从3米高的地方摔下去都没坏，质量非常好'
        lstm_predict(string)
```

运行结果：