​

基于LSTM的二分类文本情感分析

院系：计算机科学与技术

姓名：王晨懿

班级：1603104

学号：1162100102

目录

[基于LSTM的二分类文本情感分析 1](#_Toc5916916)

[1任务描述 4](#_Toc5916917)

[1.1本文工作 4](#_Toc5916918)

[1.2实验要求 4](#_Toc5916919)

[2数据说明 4](#_Toc5916920)

[2.1训练数据 4](#_Toc5916921)

[2.2测试数据 4](#_Toc5916922)

[2.3停用词数据 5](#_Toc5916923)

[3系统方法 5](#_Toc5916924)

[3.1数据预处理 5](#_Toc5916925)

[3.1.1从文件读入文本并整理为输入矩阵 5](#_Toc5916926)

[3.1.2分词 5](#_Toc5916927)

[3.1.3去除停用词 5](#_Toc5916928)

[3.2 word2vec生成词向量 5](#_Toc5916929)

[3.3建立docs2idx & idx2vec 6](#_Toc5916930)

[3.4 LSTM模型 6](#_Toc5916931)

[3.4.1建立模型 7](#_Toc5916932)

[3.4.2 配置模型 7](#_Toc5916933)

[3.4.3训练模型 7](#_Toc5916934)

[3.4.4模型评价 8](#_Toc5916935)

[3.4.5保存模型 8](#_Toc5916936)

[4结果与分析 8](#_Toc5916937)

[4.1停用词的影响 8](#_Toc5916938)

[4.2系统性能 8](#_Toc5916939)

[5总结 8](#_Toc5916940)

[参考资料 8](#_Toc5916941)

# 1任务描述

## 1.1本文工作

文本情感分类任务作为NLP的常见任务，具有很高的应用价值。本文采用LSTM模型，训练一个识别文本postive和 negative的二分类分类器。

## 1.2实验要求

设计和实现分类系统，完成对短文本的情感极性（积极、消极）二分类任务。

程序语言、框架、学习方法不限，可使用外部语料，不可使用已有的情感分析或文本分类库。

# 2数据说明

## 2.1训练数据

共有10000条训练数据，包括5000条积极情感文本（sample.positive.txt）和5000条消极情感文本（sample.negative.txt）；

文件为“UTF-8”编码，数据以xml格式存储，格式如下：

<review id="n">  
xxx  
</review>

每个“review”标签是一条训练数据，“id”是训练数据编号（0到9999），标签内容“xxx”为文本内容。

## 2.2测试数据

共有2500条未知类别（积极或消极）的测试数据，使用实现的系统对其进行预测。

文件为“UTF-8”编码，数据以xml格式存储，格式如下：

<review id="n">  
xxx  
</review>

每个“review”标签是一条训练数据，“id”是训练数据编号（0到2499），标签内容“xxx”为文本内容。

要求对测试数据进行预测，积极用“1”表示；消极用“0”表示。

## 2.3停用词数据

数据来源：<https://github.com/chdd/weibo/tree/master/stopwords>

# 3系统方法

## 3.1数据预处理

### 3.1.1从文件读入文本并整理为输入矩阵

使用re.findall() 方法从文件中获取所有的用例。

使用numpy.concatenate() 方法将正反用例整合为一个矩阵docs，矩阵的每一行为分好词的一个用例。

def get\_text(file\_path):  
 with open(file\_path, 'r', encoding='utf-8')as f:  
 text = f.read().replace('\n', '')  
 return re.findall(r'<review id="\d+">(.+?)</review>', text, flags=re.S)  
  
neg = [jieba.lcut(x) for x in get\_text(os.path.join(DATA\_PATH, 'sample.negative.txt'))]  
pos = [jieba.lcut(x) for x in get\_text(os.path.join(DATA\_PATH, 'sample.positive.txt'))]  
docs = np.concatenate((neg, pos))

### 3.1.2分词

本文使用jieba工具包对读入的每一段文本进行分词，将分词后的文本作为进一步的输入。

### 3.1.3去除停用词

首先从文件中加载停用词表。

对于分词后的文本，将分词结果中的每个词与停用词表进行比对，如果是停用词，则删除。

## 3.2 word2vec生成词向量

在分析文本的过程中，我们需要把句子用数字的形式表示出来。词语和文档的表示方法至关重要。一种很好的解决方法是用高维向量（词向量，Word Embedding）表示词语。

本文使用了Google开源的著名的工具Word2Vec方法。Word2Vec是用来产生词向量的相关模型。训练完成之后，word2vec模型可用来将每个词映射到一个固定长度向量，这些词向量可用来表示词对词之间的关系，甚至用欧氏距离或余弦相似度都可以找出两个具有相近意思的词语。

本文使用的是Python的Gensim库中的Word2Vec子库来实现。

1. 建立模型，设置生成的词向量维度为300，并且只考虑出现频率超过10的词语。

model = Word2Vec(size=n\_dim, # 300 词向量的维度  
 min\_count=n\_exposures, # 10 所有频数超过10的词语  
 window=7, workers=4)

1. 设置词汇表，docs是经过上文分词和去停用词处理过后的文本，其类型是一个list，长度为10000，即包含10000个训练用例。list中的每一个元素也是一个list，对应一个训练文本。

model.build\_vocab(docs) # input: list

1. 训练模型

model.train(docs, epochs=model.epochs, total\_examples=model.corpus\_count)

1. 保存模型，将在测试时用到

model.save(os.path.join(MODEL\_PATH, 'w2v\_model.pkl'))

## 3.3建立docs2idx & idx2vec

使用gensim库的Dictionary.doc2bow 对word2vec的词汇表建立词袋模型，这里的词汇表中只包含word2vec训练时出现次数不少于10次的词。词袋模型建立完成后可以得到每个词及其在词袋模型中的索引。

gensim\_dict = Dictionary()  
gensim\_dict.doc2bow(model.wv.vocab.keys(), allow\_update=True)

我们将这些索引拿来，替换文本矩阵中每个词，从而得到用索引表示的文本**docs2idx**。需要注意的是，对于出现频率小于10的词的索引设为0，相应的其他词的索引都要+1。

同样的方法，我们还能依据词袋模型和word2vec训练好的词向量，得到一个从索引映射到词向量的dict，命名为**idx2vec**。

另外需要使用gensim库的sequence() 方法，将docs2idx的每个句子中词的数量统一为max\_len，本文选取的长度为100。则docs2idx变为一个10000\*100的二维矩阵。

docs2idx = sequence.pad\_sequences(docs2idx, maxlen=max\_len)

## 3.4 LSTM模型

我们已经分好词，并且获得了词向量，那么句子就对应着词向量的集合

本文使用Keras库搭建LSTM模型

### 3.4.1建立模型

使用keras库的Sequential模型搭建。Sequential可以理解为多个网络层的线性堆叠。

model = Sequential()

1. 首先是Embedding层，它可以将正整数（索引）转换为固定大小的密集向量。我们规定输出的维度为300；输入的维度为词汇表的大小n\_symbols，即len(idx2vec)；权重为每个词对应的词向量idx2vec；输入序列的长度为之前为文本统一好的长度100。

model.add(Embedding(output\_dim=n\_dim, # 词向量的维度  
 input\_dim=n\_symbols, # 词汇表大小  
 mask\_zero=True, # 把 0 看作为一个应该被遮蔽的特殊的 "padding" 值, 索引 0 不能被用于词汇表中  
 weights=[embedding\_weights],  
 input\_length=max\_len)) # 输入序列的长度

1. 然后是LSTM模型，选取激活函数为sigmoid

model.add(LSTM(activation="sigmoid", # 选取激活函数  
 units=50)) # 输出维度

1. Dropout 会在训练中每次更新时， 将输入单元的按比率随机设置为 0， 这有助于防止过拟合。

model.add(Dropout(0.5))

1. 设置全连接层和激活函数

model.add(Dense(1)) # Dense=>全连接层,输出维度=1  
model.add(Activation('sigmoid'))

### 3.4.2 配置模型

对于二分类问题指定损失函数为binary\_crossentropy

优化器为梯度下降算法adam

度量标准为准确率。

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

### 3.4.3训练模型

x\_train, y\_train为训练样本及其标记；batch\_size为每次梯度更新的样本数；epochs为迭代次数。

本文设置batch\_size = 32，epochs = 4

model.fit(x\_train, y\_train,  
 batch\_size=batch\_size, # 每次梯度更新的样本数  
 epochs=n\_epoch, # 训练在样本上迭代次数  
 verbose=1) # 显示进度条

### 3.4.4模型评价

使用evaluate方法可以进行评价。

在此之前可以使用train\_test\_split方法将样本分为测试集和训练集。

x\_train, x\_test, y\_train, y\_test = train\_test\_split(docs2idx, y, test\_size=0.2)

evaluate方法输出的分数包含两个元素，第一个元素为loss的值，第二个元素为测试样本的准确率。

score = model.evaluate(x\_test, y\_test, batch\_size=batch\_size)  
print('Test score:', score)

### 3.4.5保存模型

yaml\_string = model.to\_yaml()  
with open(os.path.join(MODEL\_PATH, 'lstm.yml'), 'w') as outfile:  
 outfile.write(yaml.dump(yaml\_string, default\_flow\_style=True))  
model.save\_weights(os.path.join(MODEL\_PATH, 'lstm.h5'))

# 4结果与分析

## 4.1停用词的影响

## 4.2系统性能

# 5总结

本文是参考<https://github.com/BUPTLdy/Sentiment-Analysis> 实现的。很多方法都是第一次见到，查询手册才能一步步弄明白。

本次任务使用word2vec模型和LSTM模型，这二者在之前的学习中有过了解，然而具体使用还是第一次。总的来说收货很大，也是第一次实现一个情感分析系统。

# 参考资料

<https://github.com/BUPTLdy/Sentiment-Analysis>

<https://keras.io/>