華東理工大學

模式识别大作业

题目：	句子情感分类
学院：	信息科学与工程学院
专业：	控制科学与工程
姓名：	孙骏
学号：	Y30190745
指导老师：	赵海涛

完成日期： 2019 年 12 月 12 日

经过数周的学习，在赵老师的指导与激励（上课不仅仅是鼓励同时也会以身说法变相的激励）下，我对模式识别以及机器学习有了一点浅显的认识，希望对所学的知识加以认知，不仅仅停留在表面，辅以题目进行锻炼，增加自己的实践能力。

1. 句子情感分类

经过对题目的寻找，分析，以及感兴趣的程度，最终确定对 lintcode 上的句子情感分类问题进行学习与处理。

情感分类问题是一个简单的自然处理的题目，根据所提供数据进行分析，题目中所有的句子全部来源于社交网络和博客，是带有感情色彩的主观性文本。本题目的在于区分一句话的情感是“积极的”还是“消极的”，其中积极情感的标签为 1，消极情感的标签为 0，利用给定数据集进行训练，从而对测试集中的句子进行情感判断，并给出相应标签。

2. 整体解决方案

1）对数据进行分析

2）构造词向量

3）通过对训练集进行训练，建立相关模型，进行分类

4）对测试集的数据进行处理，输入，得到响应的输出

2.1对数据进行分析

	A	B
1	sentence	label
2	Ok brokeback mountain is such a horrible movie.	0
3	Brokeback Mountain was so awesome.	1
4	friday hung out with kelsie and we went and saw The Da Vinci Code SUCKED!!!!!	0
5	I am going to start reading the Harry Potter series again because that is one awesome story.	1
6	Is it just me, or does Harry Potter suck?...	0
7	The Da Vinci Code sucked big time.	0
8	I am going to start reading the Harry Potter series again because that is one awesome story.	1
9	For those who are Harry Potter ignorant, the true villains of this movie are awful creatures called dementors.	0
10	Harry Potter dragged Draco Malfoy 欹?s trousers down past his hips and sucked him into his throat with vigor, ma	0
11	So as felicia's mom is cleaning the table, felicia grabs my keys and we dash out like freakin mission impossible.	1
12	I love The Da Vinci Code...	1
13	I love Brokeback Mountain.	1
14	The Da Vinci Code sucked big time.	0
15	Harry Potter is AWESOME I don't care if anyone says differently!..	1
16	I LOVED BROKEBACK MOUNTAIN!..	1
17	stupid brokeback mountain.	0
18	We liked Mission Impossible.	1
19	friday hung out with kelsie and we went and saw The Da Vinci Code SUCKED!!!!!	0
20	And this is why I hate Harry Potter.	0
21	I want to be here because I love Harry Potter, and I really want a place where people take it serious, but it is still sc	1
22	He's like 'YEAH I GOT ACNE AND I LOVE BROKEBACK MOUNTAIN.'	1

从文件中可以看到，数据比较简单，sentence 以及其对应的 label，数据集的数据量也比较充足，一共有 5669 句话，测试集也是这种形式，只是没有响应的 label，需要预测后，输入。

2.2构造词向量

将数据引入 python 中，并进行相应的处理

```
def load_data(filepath, input_shape):
    df = pd.read_csv(filepath)
    labels, vocabulary = list(df['label'].unique()), list(df['sentence'].unique())
    string = ''
    for word in vocabulary:
        string += word

    vocabulary = set(string)
    word_dictionary = {word: i+1 for i, word in enumerate(vocabulary)}

    with open('word_dic.pk', 'wb')as f:
        pickle.dump(word_dictionary, f)
    inverse_word_dictionary = {i+1: word for i, word in enumerate(vocabulary)}
    label_dictionary = {label:  i for i, label in enumerate(labels)}
    with open('label_dic.pk', 'wb')as f:
        pickle.dump(label_dictionary, f)
    output_dictionary = {i: label for i, label in enumerate(labels)}

    vocab_size = len(word_dictionary.keys())  # 词汇表大小
    label_size = len(label_dictionary.keys())  # 标签大小
```

首先，引入数据，并将 sentence 列中值放于 vacabulary 中，label 列的值放在 label 中，而后提取 vacabulary 其中的值，这里主要是提取出所有句子中涉及到的字母以及标点，放入一个字典中，构成接下来使用的一个简单字典，label 进行同样的处理。然后，利用 keras 中的 embedding，构造词向量——构

```
model.add(Embedding(input_dim=vocab_size+1, output_dim=output_dim, input_length=input_shape, mask_zero=True))
```

造的简单的词典中，字母存在相应的序号，通过 embedding，将整数行的序列构造成响应的浮点型的词向量，同时进行降维处理。

其中，input_dim 的值是词汇表的大小，由于 make_zero 的值真，所以索引 0 不能够用到词汇表中，所以 input_dim 需要是词汇表的值加 1，output_dim 是输出的维数（自己设定）。

2.3建立模型

利用 keras 中的 lstm 进行相关的参数的输入，建立模型。

```
model.add(LSTM(n_units, input_shape=(x.shape[0], x.shape[1])))
model.add(Dropout(0.2))
model.add(Dense(label_size, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

对模型利用数据对模型模型进行训练。

```
# 模型训练
lstm_model = creat_LSTM(n_units, input_shape, output_dim, filepath)
lstm_model.fit(x, y, epochs=epochs, batch_size=batch_size, verbose=1)

lstm_model.save(model_save_path)
```

其中，x 是将 sentence 每一行的值进行了序列上的转换，在进行了词向量的构建后，每句话就转换成了由词向量构成矩阵，然后进行有监督地训练。

2.4测试集测试

将测试集的 sentence 导入，调用已经建立好的词典以及 LSTM 模型，将测试集需要检测的 sentence 值输入，得到相应的输出，二者调整格式，输出 csv 文件。这里主要利用了 for 循环，分别对每一行进行输入，得到相应输出，而后利用 python 中内置的 csv，将结果输出，代码如下。

对测试集数据进行处理（其中该句子中有可能出现未曾列入词典的符号，无法进行预测，则输出没有相应符号）

```
df = pd.read_csv('C:\\Users\\cainiao\\Desktop\\test.csv')
data = []
num = []
lstm_model = load_model('./sentiment_analysis.h5')
for sent in df['sentence']:
    try:
        input_shape = 50
        x = [[word_dictionary[word] for word in sent]]
        x = pad_sequences(maxlen=input_shape, sequences=x, padding='post', value=0)
        y_predict = lstm_model.predict(x)
        label_dict = {v: k for k, v in output_dictionary.items()}
        label_predict = label_dict[np.argmax(y_predict)]
        data.append(label_dict[np.argmax(y_predict)])
        num.append(sent)
        # print(label_dict[np.argmax(y_predict)])
    except KeyError as err:
        a = '没有相应符号'
        data.append(a)
        num.append(sent)
```

输出带有预测的测试集

```
c = list(zip(num, data))
print(c)


with open('submission.csv', 'w',errors='ignore') as csvFile:
    writer = csv.writer(csvFile)
    writer.writerow(('sentence', 'label'))
    writer.writerows((c))
```

3. 总结

虽然成功的完成了对测试集的预测，但由于初次接触自然语言处理，但整个处理思路以及处理方法上有很大的不足。

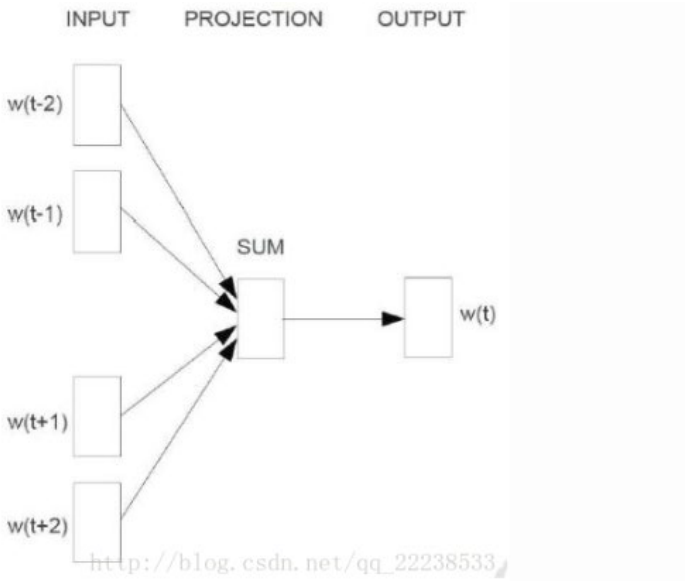
1）没有对训练数据进行合理的分词，本文是对字母以及符号进行了划分，相较于英文中以空格进行分词，这样的想法虽然不会存在这种情况（因为没有新的句子中存在没有训练过的单词，造成预测的偏差），但是，对于训练模型中所用到的句子 x 来说，能够构建的训练以及测试的信息太少，同样由 50 个词向量构成的句子，本文是 50 个字母构成的句子，而对单词进行分割的方法是 50 个单词构成的句子，所以相当大程度上，造成了测试不准确。

2）没有提前对数据集进行预处理，使用 keras 本身的 embedding 可以构造词向量，但是没有引入外部信息，也就是没有考虑到词与词之间的关联性，因而，应该利用 word2vec 进行词向量的构建，word2vec 中利用到的 CBOW 和 Skip-gram 模型可以构造出考虑到文本中词语相关性的词向量（见附录）。而后，再输入到 embedding 中。

3）由于时间关系以及能力不足，没有实现对比的操作，该题其实在构建出词向量后可以看作是二分类问题，可以利用课堂上所学习的知识进行分类与利用 LSTM 处理分类的结果进行对比。

经过这次实验，虽然由于初次接触自然语言处理知识，整个过程比较艰难，也存在诸多不足，但对模式识别的认识不是仅停留于表面，同时也算是完成了一个自己的作品，这里要感谢赵老师的教学，训练了我处理相关问题的思维与能力，为未来的选择多提供了一个方向。

附录



CBOW 语言模型

简单来讲，词向量的构建就是将单词以映射到一个新的空间中，以多维的向量来表示，词向量从原始的稀疏表示过度到低维空间中的密集表示。在用稀疏表示时通常会出现维数灾难的问题。比如经典的 one-hot 编码，[0,0,0,0,1,0,0,0…]表示词典中的单词，通过这样的方式来表示词典中的单词，但是这种表示在单词很多的情况下，会造成维度灾难的问题，同时我们可以看到两个词向量之间是不相关的，所以合适的词向量的构建对于分析词与词之间的关系有着十分重要的作用，因此，在构建网络模型之前对单词进行预处理是十分必要的。

目前，大家主要利用 Word2vec 工具得到词向量。Word2vec 工具中主要包含两个模型：CBOW 和 Skip-gram。简单介绍 CBOW ，该模型就是利用一个词所在的上下文中的词作为输入，而这个词本身作为输出，不断迭代，满足构造的误差函数，而所需要的词向量就是过程中的权重矩阵W_{N*M}。Skip-gram 则是与之相反。