《语言分析与机器翻译》课程实践报告

题　　目 基于 LSTM 网络的中文文本情感分析

姓　　名 赵雄雄

学　　号 2101863

2021 年 11 月

摘　要

社交网络的普遍，带来了情感和观点的宣泄和爆炸。根据情感进行心理健康的监控和舆论走向的把控，都使情感分析在现今社会具有重要意义。而本实验采用深度学习中的 LSTM（长短期记忆网络）来实现对中文对话文本中的情感分析，最终实现对情感的分类及预测。具体实现方面采用高质量的六情感微博数据为基础，利用 Jieba 进行分词，将中文文本序列化，同时进行词嵌入和文本填充，再使用 LSTM 算法进行核心的分类器训练，调整模型实现最优的准确率。最后自定义一个预测函数，通过输入一段中文文本来预测文本的情感。

关键词：情感分析；词嵌入；LSTM；深度学习

ABSTRACT

The prevalence of social networks has led to the catharsis and explosion of emotions and opinions. According to the monitoring of mental health and the control of the trend of public opinion, emotional analysis is of great significance in today's society. In this experiment, LSTM (long- and short-term memory network) in deep learning is used to realize the emotional analysis of the text of Chinese dialogue, and finally to classify and predict emotion. Based on high-quality six-emotion microblog data, Jieba is used to serialize Chinese text, while embedding words and filling text, and then using LSTM algorithm to train the core classifiers, adjusting the model to achieve optimal accuracy. Finally, customize a prediction function to predict the emotion of the text by entering a piece of Chinese text.

Key words： Emotional analysis; word embedding; LSTM; deep learning

1. 引言

随着计算机应用的普及和互联网技术的发展，博客、电子邮件、论坛、聊天室、在线评论等互联网应用无处不在，越来越多的用户通过互联网发表带有观点、表达情感的文字、图片或者表情符号等。在这些带有人们观点的信息通过清洗、特征提取等操作提取出大量有用丰富的信息，在这些海量信息中，文本信息数据量最大且较容易获得，因此基于文本的情感分析已经成为人工智能的一个热门且前沿的研究领域。

文本情感分析自 2002 年被提出之后，就受到广泛关注并取得了一定成就，特别是在在线评论的情感倾向性分析上获得了很大的发展。情感分析也逐渐经历了由粗到细、由简到繁的过程，与此同时文本情感分析的应用也从简单的了解用户喜好扩展到为个人、企业及机构提供决策支持，网络舆情风险分析，信息预测等。情感分析是一种常见的自然语言处理(NLP)中最活跃的研究领域之一，它是对带有情感色彩的主观性文本进行分析、处理、归纳和推理，利用一些情感的指标来量化定性数据的方法。

现在主流的情感分析方法有两种:一种基于词典的方法，一种是基于机器学习算法的方法。基于词典的方法主要通过制定一系列的情感词典和规则，对文本拆解、关键词提取，计算情感值，最后通过情感值来作为文本的情感倾向依据。而基于机器学习的文本情感分析需要大量的人工标注的语料作为训练集，通过提取文本特征，构建分类器来实现情感的分类。

在近几年，随着深度学习的应用，深度学习在传统特征选择与提取框架上取得巨大突破，在自然语言处理、生物医学分析、遥感影像解译等诸多领域产生越来越重要影响，并在计算机视觉和语音识别领域取得革命性突破。本文选择长短期记忆模型（LSTM）。一种基于时间序列的深度学习模型。它是在循环神经网络(RNN)基础上增加了“门”的控制，使其很好的处理长期依赖问题，加上基于时间序列模型的方法，所以我认为更适合处理自然语言，会取得更好的效果。

2. 研究内容概述

为了更有效地进行情感分析，本实验采用深度学习中的长短期记忆网络（LSTM）来进行建模，实现对情感的分类及预测。具体实现方面本实验以高质量的六情感微博对话数据为基础，去停用词并利用 Jieba 进行分词，然后将中文文本序列化，同时进行词嵌入和文本填充，向量化后再使用 LSTM 算法进行核心的

分类器训练，调整模型实现最优的准确率。最后我还自定义一个预测函数，通过输入一段中文文本来预测文本的情感。

3. 数据介绍及预处理工作

3.1 数据介绍

本实验所用数据包含两个维度，一是中文的文本内容 text，主要来自微博上的对话内容。二是是根据对话内容所打的情感标签，主要包含 6 大类，分别是 Happiness、Sorrow、Fear、Disgust、None 和 Love。而我们的任务就是挖掘出训练数据集中的语义特征并对测试集进行分类预测。原始数据如下图所示：

A	B	C
text	label	
花少里也是这样说郑爽，其馀几人	None	
恐龙君，帮忙介绍一下咯	None	
那我觉得蓝色那系列应该没问题，夏天或许会油	None	
这个软件感觉好赞，还是你p图技术比较高啊不管了我也要去下个试	Love	
期待他们同台的一天，我也就心满意足了	Love	
哈哈是吧他一直这么洒从未被超越	Happiness	
实在闲的不行了。我就一天打一两把。目标是打够了人机再去对战	None	
我一开始也纳闷了，居然没人说念破的事了	None	
好么？！白了么？	None	

但是想要获得良好的预测结果，进行有效的数据预处理十分必要的。为此，我们做出以下工作。

3.2 数据处理

3.2.1 缺失值处理

首先我们对数据集进行缺失值查看，发现数据中并不存在缺失值。

In [5]:
```
print("在 Text 列中总共有 %d 个空值." % df['Text'].isnull().sum())
print("在 lable列中总共有 %d 个空值." % df['label'].isnull().sum())
df[df.isnull().values==True]
df = df[pd.notnull(df['label'])]
```

```
在 Text 列中总共有 0 个空值.
在 lable列中总共有 0 个空值.
```

3.2.2 去停用词和分词处理

接下来我们进行去停用词。因为我们知道中文的对话文本内容中包含很多日常使用频率很高的常用词,如吧、吗、呢、啥等一些感叹词等。这些词和符号对系统分析预测文本的内容没有任何帮助,反而会增加计算的复杂度和增加系统开销,所以在使用这些文本数据之前必须要将它们清理干净。所以我们进行去停用词操作。

下一步进行分词操作。因为一段文本中往往有不止一个情感词，这些情感词对文本所要表达的情感起着非常关键的作用,有时候文本中的某一个情感词就决定了整段文本的情感,因此我们在进行情感分析之前就需要把这些情感词都找出来，那就必须先把文本分成词语的序列，也就是分词。对文本进行有效分词是进行情感分析之前一个非常重要的基础步骤。对于中文文本，句子与句子之间用句号隔开，但是句子内的词与词之间却没有明显的符号分隔，因此中文文本的分词难度要高于英文文本的分词。就目前来说，中文分词的方法主要有三种：基于理解的分词、基于统计的分词、基于字符串匹配的分词。目前比较成熟的分词服务主要有：阿里巴巴研发的阿里云 NLP,腾讯研发的腾讯文智，复旦大学自然语言处理组研发的 FudanNLP,哈工大社会计算与信息检索研究中心研发的 LTP，以及在 python 语言中最流行的分词工具结巴分词等。其中结巴（jieba）分词完全开源，分词功能具有效率高、准确率高的特点，因此本文最终选择结巴（jieba）作为分词工具。

进行这些初步的数据预处理之后我们得到的数据如下图所示：

	A	B	C
10	冷酷 贱 哈哈哈 萌	Happiness	
11	最 好喝 芋头 风暴 牛奶 冰 夏天 吃 比较 爽	Love	
12	脸太肉 需要 遮起来 哈哈哈	Happiness	
13	每次 帮 身边 收尾 奶茶 朋友 不错 收尾 杯 真的 会 撑	None	
14	越多人 搭理 越 上劲	Disgust	
15	川魂 冒牌货 冒菜 很 好吃 吃 好	Love	
16	不用 每去 尤安店 回家 感叹 诶 心里 头 晓得 行赖	None	
17	抱住 我丁 委屈 不行	Sorrow	
18	喜欢 很 喜欢	Love	
19	哈哈哈 哈哈 再 熬 一周	Happiness	
20	不好 笑 见	Fear	
21	相比之下 会 好 很多 我家 这位 没个 停 希望 有个 女儿 妹妹	Love	
22	哈哈 说明 爱惜	Happiness	
23	关注点 总是 很 奇怪 哈哈哈	Happiness	
24	老夏 家 真的 超级 温馨	Love	
25	小本经营 超薄 利 嘤 嘤 嘤	Sorrow	

3.2.3 序列化和词嵌入

但是我们并不能直接把中文文本带入模型，所以我们要把中文文本进行序列化操作和词嵌入操作。

我们要将 text 数据进行向量化处理，我们要将每条 text 转换成一个整数序列的向量。本文选择 keras 的分词器（Tokenizer）进行序列化。使用 Tokenizer 根据数据集生成词典，并且将词典保存为.json 文件，以便预测的时候可以再次使用该词典。如下图所示，词典中每个词存在且仅存在一个索引与之对应，并且 Tokenizer 还会统计数据集中各个词出现的频数。然后通过 texts_to_sequences（）方法将文本序列化，也就是将文本转换成由索引构成的序列。

词汇	转换值
“默默地”	3390

接下来的操作就是词嵌入。因为要想让计算机理解文字的含义，就必须用某种方法将计算机无法理解的文字内容转化为计算机可以理解的数字、矩阵向量、符号，使其在计算中获取信息。为了解决此问题，我们引入词嵌入，词嵌入的本质上是一个映射，它可以将词语或者短语映射到一个高维度的空间中形成词向量，如果有两个词的词义是相近的或者相同的，那么这两个词在映射后他们在空间中的距离也是接近的。目前比较流行的词嵌入主要有：谷歌在新闻数据集上训练出来的 word2vec、斯坦福大学提出并且训练的 GloVe。中文领域的词嵌入相对较少。在国内，腾讯人工智能实验室公布了特定于中文领域的词嵌入映射文件 Tencent_AILab_ChineseEmbedding.txt，本文件将中文词汇分别映射到多维空间中，本词嵌入文件不论在词汇覆盖率还是在映射后的效果都很高，因此本文选择 Tencent_AILab_ChineseEmbedding.txt 进行词嵌入。由于需要对这么多个词中的每一个都映射为多维的向量并存储到文件中，这必然导致最后的文件占用的内存非常的大，解压出来的腾讯词嵌入文件大小达 16.0G，而本计算机的内存只有 8G，显然无法直接将文件载入内存中。为了解决此问题，本文专门创建了词嵌入文件的映射文件：embeddings_map_index.txt，该文件存储的是原词嵌入文件的索引，也就是只保留原词嵌入文件中的词汇字段、词汇在词嵌入文件中的起始位置、词汇在词嵌入文件中占用的长度。只要根据本映射文件就可以快速的从原词嵌入文件中读取需要的词向量，而不需将整个词嵌入文件读入内存。（注：由于 Tencent_AILab_ChineseEmbedding.txt 和 embeddings_map_index.txt 两个文件太大，故未在 GitHub 上面上传，有需要可以在腾讯人工智能实验室官网下载使用。）

具体代码如下：

```
EMBEDDING_DIM = 200
emb = ReadEmbeddings()
emb_map_list = emb.load_map_in_memery()
embedding_matrix = np.random.random((MAX_NB_WORDS+1, EMBEDDING_DIM))
for word, i in word_index.items():
      query_word = word.replace("'", "").strip()
      embedding_vector = emb.find_by_word(emb_map_list, query_word, default_value=[0.0]*200)
      if embedding_vector != [0.0]*200:
            embedding_matrix[i] = embedding_vector
```

3.2.4 文本填充

我们可知道将数据输入网络之前需要先对数据序列进行均匀地切分。由于文本的长度不一，因此不同文本序列化后的向量长度也是不同的，这样就不能进行统均匀地切分。为了解决这一问题，我们需要进行文本填充，先指定填充长度 MAXLEN，若序列化后的文本向量长度小于 MAXLEN，那么就用“0”进行填充，直到长度文本向量长度刚好为 MAXLEN。若序列化后的文本向量长度大于 MAXLEN，那么就将超出的部分截断，只能保留长度为 MAXLEN 的文本向量。

代码如下：

```
X = tokenizer.texts_to_sequences(df['Text'].values)
#填充X,让X的各个列的长度统一
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)

#多类标签的onehot展开
Y = pd.get_dummies(df['label_id']).values

print(X.shape)
print(Y.shape)
```

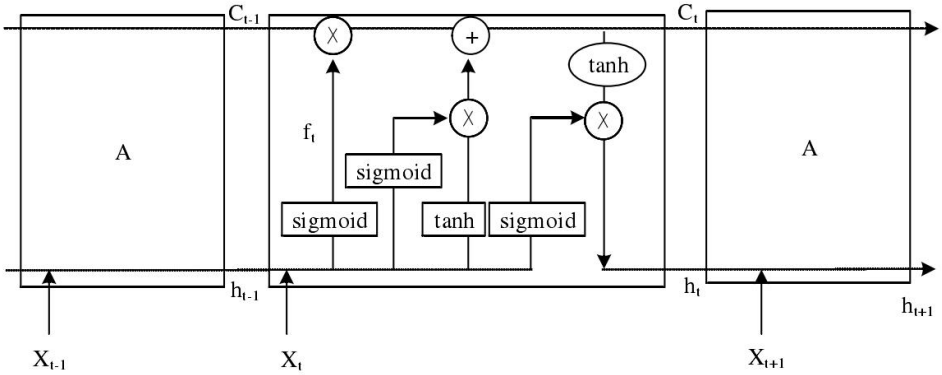
```
(29448, 30)
(29448, 6)
```

4. 模型搭建与训练

4.1 LSTM 网络介绍

长短期记忆网络（LSTM，Long Short-Term Memory）是一种时间循环神经网络，是为了解决一般的 RNN（循环神经网络）存在的长期依赖问题而专门设计出来的。且 LSTM 适用于解决带时间序列的数据预测问题。

LSTM 神经网络的结构与循环神经网络相似，将循环神经网络的循环结构替换成了自身独有的网络结构。LSTM 神经网络的循环网络结构同样可以看作在时间维度上的不断展开，每一个重复模块都代表了该模块在当前时间上的状态。其循环结构内块包含一个单元状态(cell state)、四个神经网络层和三个门结构三部分。LSTM 神经网络结构如下图所示。



LSTM 神经网络的重复结构中最上方的水平线代表单元状态，它贯穿了循环结构，是整个网络信息持久记忆的关键。每个重复模块的输入为上一个记忆模块的短时记忆 ht-1、当前记忆模块的输入 Xt 和神经网络的长时间记忆状态 Ct-1 三部分；输出为当前时刻产生的长期记忆状态 Ct 和工作状态 ht。

四个神经网络层分别为三个单层的 sigmoid 前馈神经网络层和一个单层的 tanh 前馈神经网络层。

三个门结构分别为输入门、输出门和遗忘门，是 LSTM 神经网络能够筛选信息，决定信息是否记忆或遗忘的关键。每个门结构分别由一个 sigmoid 神经网络层和一个向量操作构成。神经网络层的 sigmoid 激活函数用于调节在神经网络中的输出数据，控制输出数据在 0 到 1 之间，防止数据通过神经网络进行计算后出现剧烈变化导致最终不可控制。因此，门控机制采用 sigmoid 层形成输入数据的丢弃或通过机制。

其中，遗忘门负责决定舍弃多少上一时刻的单元状态的信息。遗忘门的 sigmoid 层通过读取上一时间状态的短时记忆输出信息 ht-1 和当前时间状态新的输入信息 xt，经过分析决定上一时刻可以通过的状态信息量，生成一个在 0 到 1 范围内的控制值 ft。计算公式如式所示：

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{1}$$

输入门负责决定保留多少新的输入信息到上一时刻的的单元状态中。它分为保留新输入信息和向量转化两步，第一步上一时刻短时记忆输出信息 ht-1 和当前时刻新的输入信息 xt 通过输入门的 sigmoid 层，生成 it 值，决定保留多少新信息，如式(2)所示；第二步将输入信息 ht-1 和 xt 通过 tanh 层神经网

络，将其转化为候选状态 ct，用来更新上一时刻单元状态，如式(3)所示。

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (2)$$

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_i) \quad (3)$$

信息经过遗忘门和输出门的筛选后，可以对上一时刻的单元状态进行更新，如式(4)所示。其中遗忘门控 ft 值与上一时刻单元状态 Ct-1 按位相乘，表示需要从上一时间状态删除的信息，输入门控值 it 和候选状态 tC2 按位相乘表示将要向上时间状态增加的信息，二者相加后将上一单元状态更新为当前单元状态 Ct。

$$C_t = f_t * c_{t-1} + i_t * \tilde{c}_t \quad (4)$$

输出门决定输出多少当前的单元状态的信息。首先同样将输入信息 ht-1 和 xt 通过 sigmoid 层，生成 ot 值，决定多少当前单元状态的信息将被输出，如式(5)所示；然后当前单元状态的信息经过 tanh 层处理，将值控制在-1 到 1 之间，再与输出门控值 ot 按位相乘得到当前状态将要输出的信息，如式(6)所示。

$$O_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (5)$$

$$h_t = o_t * \tanh(C_t) \quad (6)$$

4.2 模型搭建

模型的第一次是嵌入层(Embedding)，它使用长度为 100 的向量来表示每一个词语。SpatialDropout1D 层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。LSTM 层包含 100 个记忆单元。输出层为包含 6 个分类的全连接层。由于是多分类，所以激活函数设置为'softmax'，损失函数为分类交叉熵 categorical_crossentropy。

具体代码如下：

In [13]:
```
#定义模型
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(6, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

运行结果如下：

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 30, 100)           5000000

 spatial_dropout1d (SpatialD  (None, 30, 100)          0
 ropout1D)

 lstm (LSTM)                 (None, 100)               80400

 dense (Dense)               (None, 6)                 606

=================================================================
Total params: 5,081,006
Trainable params: 5,081,006
Non-trainable params: 0
_________________________________________________________________
```

4.3 模型训练

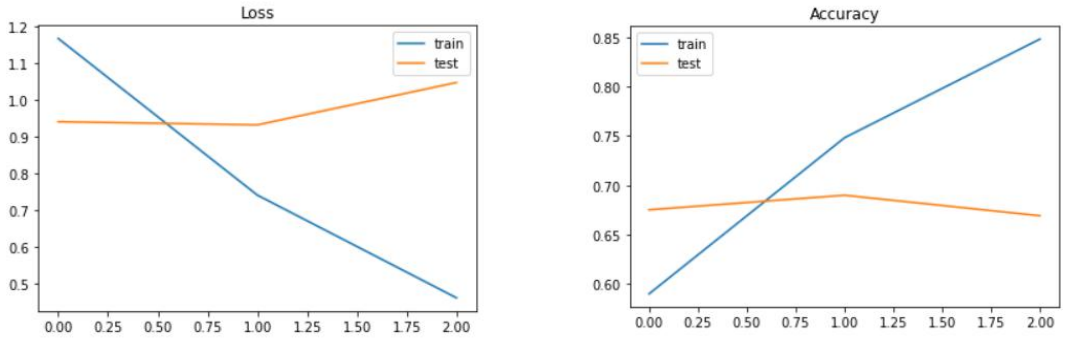
搭建好模型之后我们就需要对模型进行训练，开始我们设置 3 个训练周期，并定义好一些超参数（后边调优时会对参数进行微调）。

```
In [14]: epochs = 3
         batch_size = 64

         history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,validation_split=0.1,
                             callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)])
```

```
Epoch 1/3
373/373 [==============================] - 21s 51ms/step - loss: 1.1679 - accuracy: 0.5901 - val_loss: 0.9407 - val_accuracy: 0.6752
Epoch 2/3
373/373 [==============================] - 18s 48ms/step - loss: 0.7404 - accuracy: 0.7482 - val_loss: 0.9320 - val_accuracy: 0.6899
Epoch 3/3
373/373 [==============================] - 17s 47ms/step - loss: 0.4601 - accuracy: 0.8481 - val_loss: 1.0476 - val_accuracy: 0.6692
```

同时我们可以画出模型的准确率和 Loss 函数变化曲线，如下图所示：



从上图中我们可以看见，随着训练周期的增加，模型在训练集中损失越来越。准确率越来越高；而在测试集中，损失随着训练周期的增加由一开始的从大逐步

变小,再逐步变大。准确率随着训练周期的增加由一开始的从小逐步变大，再逐步变小。

4.4 模型评估

多分类模型一般不使用准确率(accuracy)来评估模型的质量,因为accuracy不能反应出每一个分类的准确性,因为当训练数据不平衡(有的类数据很多,有的类数据很少)时，accuracy不能反映出模型的实际预测精度,这时候我们就需要借助于F1分数、ROC等指标来评估模型。而本文正是采用F1分数作为评估标准，评估结果如下图：

accuracy 0.6709677419354839

	precision	recall	f1-score	support
Love	0.59	0.58	0.58	520
None	0.71	0.77	0.74	1468
Sorrow	0.43	0.28	0.34	198
Disgust	0.50	0.46	0.48	305
Happiness	0.81	0.82	0.82	428
Fear	0.33	0.04	0.07	26
accuracy			0.67	2945
macro avg	0.56	0.49	0.50	2945
weighted avg	0.66	0.67	0.66	2945

从以上F1分数上看,各类预测并不是很均衡，"None"类的F1分数最大(74%)，“Fear”类F1分数却出奇的差，我的想法是可能是因为“Fear”类的训练数据最少只有20多条,使得模型学习的不够充分,导致预测失误较多。当然，对于参数的微调也是有待进一步提高。

5. 自定义预测函数

最后我们自定义了一个预测函数,通过输入一段中文文本内容就可以实现对于文本情绪的预测，示例如下：

```
def predict(text):
    txt = remove_punctuation(text)
    txt = [" ".join([w for w in list(jb.cut(txt)) if w not in stopwords])]
    seq = tokenizer.texts_to_sequences(txt)
    padded = pad_sequences(seq, maxlen=MAX_SEQUENCE_LENGTH)
    pred = model.predict(padded)
    label_id= pred.argmax(axis=1)[0]
    return label_id_df[label_id_df.label_id==label_id]['label'].values[0]
```

```
In [36]: predict('我真的很开心呢！')
Out[36]: 'Happiness'
```

6. 总结与反思

通过本次实验的学习，我对于中文文本数据的处理能力有一定提高，对去停用词，分词，中文文本序列化和词嵌入以及文本填充几个方面有了比较熟练地掌握。其次，通过对 LSTM 网络的搭建以及训练过程，也让我逐渐熟悉了长短期记忆网络的算法思想和用法，总的来说，这次学习还是收获蛮多的。但是最终的预测结果并不是很理想，除了我想到的数据分布不均匀导致的训练不充分问题和调参的问题之外我觉得可能模型算法偏简单，需要进一步优化，能够更加充分去挖掘数据的的深层意义，这也将是我下一段 nlp 学习目标，去熟练掌握更多的像 bert 等模型。