基于电影影评的评分预测系统

蔡建伟
Automation Dept.
Shanghai Jiao Tong University
Shanghai, Minhang
steinstraveler@sjtu.edu.cn

刘启明
Automation Dept.
Shanghai Jiao Tong University
Shanghai, Minhang
liuqiming666@sjtu.edu.cn

严威豪
Automation Dept.
Shanghai Jiao Tong University
Shanghai, Minhang
ywh926934426@sjtu.edu.cn

***Abstract*—本文主要研究自然语言处理中的情感分析，我们试图通过建立一种模型，使得程序能够理解语言中的情感倾向，并应用在豆瓣电影短评的星级评估上。整个项目中我们尝试了多种方法，如朴素贝叶斯、词嵌入、长短记忆模型等，最终构建出了一种能够将电影短评内容映射到电影评定星级的网络。在测试集上测试精度时，取得了较为满意的效果。**

***Index Terms*—自然语言处理，影评情感分析，朴素贝叶斯，词嵌入，LSTM**

I. 背景介绍

当今，自然语言处理（NLP）及语言情感分析成为了机器学习和人工智能领域的研究热点，它研究能实现人与计算机之间用自然语言进行有效通信的各种理论和方法，在商业、工业和军事中有着广泛的应用前景。当下，很多前沿公司的 NLP 实现形式，基本是基于海量搜索的对话式 AI，由于这种 AI 在实际的环境中可以完成人类所下的指令，所以有着非常广泛的应用。

本项目旨在分析人类语言所蕴含的情感（喜好或者厌恶等），通过主观的打分间接实现对情感的提取与分类，以期能够在语言上挖掘人类情感的表达方式，并最终希望能够进一步理解自然语言处理这个热门研究领域。

豆瓣是我们本项目使用数据的主要来源。作为国内著名的电影评论平台，它拥有大量国内上映电影的优质电影评论。一般而言，每一条短评都有一个 1 到 5 星的评分以及一段文字表达，表示每一个短评撰写者对电影的评价及评分。可以认为，评分反映该用户对这部电影的好恶，文字表示则是对这种好恶的详细表达。如何理解一段文字，从而从中理解说话人的情感倾向，正是自然语言处理中情感分析的研究内容。我们将立足于豆瓣这一个数据量巨大的网络平台，通过对大量样本的分析与学习，得到一种电影评论到星级的映射关系，在用户任意输入一段评论之后，程序可以通过这段评论内容自动给出一个最可能的星级。

本文中，我们将会首先介绍数据获取及数据集制作方面的工作，随后介绍我们实现上述功能所用到的三种机器学习算法，以及我们在数据可视化方面所做的工作，最后对本项目做了一个简短的总结。

II. 数据爬虫及评论获取

要想进行后期的研究，数据是必不可少的。我们通过爬虫获取数据的基本思路大致如下。首先选择 60 部科幻电影和 60 部动作电影，制作成电影名与电影网页序列号的 csv 文件。随后对每一部电影分别爬取“好中差”三类各 500 条，共 1500 条短评和评分信息。最后将所有的短评与评分信息制作成 csv 文件以便后续对数据集的预处理工作。

在进行数据爬取的过程中，我们遇到了诸多问题。首先是数据量不足的问题：由于豆瓣本身的保护机制和防爬虫机制，在未登录时，每一部电影最多只能爬取 200*3=600 条短评与评分信息，这样就无法获得足够训练的数据；如果模拟登陆，豆瓣也会以验证码的方式对访问进行识别，防止机器的自动访问。另外，由于不同主题系列的电影有着不同的情节背景及人物设定，用户评论时受这些因素的影响较大，最终的影评风格差异性也大。还有一些用户在评论时，只有短评没有评分，或者评论中带有 emoji 表情、颜文字等非法字符。

我们通过大量的尝试和资料收集，上述问题最终得以解决。对豆瓣自身的防爬虫机制，我们使用了 post 模拟登陆的方法。登陆上豆瓣账号后，一部电影的好中差评都可以在相应的界面查看到 500 条，这对我们数据量的增长有很大的帮助，在解决这个问题上也花了我们许多时间。第一步，根据登陆界面的需求信息构建 formData 并填入相应的登陆信息；第二步，尝试进行 post 登陆并检测网页上是否有验证码，若无验证码说明登录成功可直接进入三层 for 循环的信息爬取；若有验证码，则将验证码保存并显示在交互界面，进行提示输入，正确输入后再进行 post 登陆即可成功。针对不同电影评论风格不同的问题，我们选择了情节关联性较强，用户评论客观性较强的 60 科幻电影和 60 部动作电影作为我们的影评爬取对象。最后对某些用户只评论不评分的问题，我们在后续的数据预处理中进行了筛除。

通过上述方式，我们成功从豆瓣上获得了约 19 万条短评和评分的初步文字数据。

III. 基于词性质的朴素贝叶斯分类法

A. 基本思路

在探究电影评论与相应星级的对应关系中，我们发现评论中的特定词语有明显的情感表达，在不同星级的评论中有明显的分布区别。比如某些褒义词语，如“紧张”、“好看”等，在高分评论中出现的次数明显多于在低分评论中出现的次数，这是因为评论者描述自己喜欢看的电影时更倾向于使用这些褒义词语。反之，一些贬义词语，如“无聊”、“睡着”等，在低分评论中更加频繁地出现。通过对评论中出现的词语进行统计，了解词语在不同星级评论中出现的频率特征，我们就能大致了解单个词语的性质。因此，在一段评论中出现的词语可以看作这段评论的特征。在输入测试评论时，通过考察其中出现的关键词语的历史星级信息，我们就能推测出该条评论的最可能星级。

在实际操作的过程中，由于词语情感的特殊性，某些词语在不同语境下可能同时具有多种含义。例如，“傻瓜”

可能是评论人观影后一种褒义的情感表达，也可能是一种不满的贬义表达。如果我们严格按照朴素贝叶斯的分类方法，把这个词严格赋予历史出现频率最高的类（如 4 星），那么就不能充分体现该词在贬义语境下的含义。我们想到的替换方法是，统计该词语在所有样本中的平均分值，将平均分值带入到测试样本的计算中。由于采用平均分值时充分考虑了所有训练样本的话语权，因此在实际的电影评论中更具可行性。

整个分类实现方法流程简述如下。首先利用中文分词库将每个训练样本（评论）中的关键词语提取出来，在此过程中根据词性进行一些筛选，防止无关词语（如人称代词“我”、中性名词“电影”等）出现在词列表中影响后续评判。统计每一个词在训练集中出现的次数、平均星级，并存储在.csv 文件中以备后续使用。在测试时，我们导入.csv 文件中每个词的平均星级。将测试评论交由中文分词库进行分词并输出每个词语的语义权重，将这些权重进行 Laplace smoothing 平滑处理并进行归一化。最后将每个词的训练平均星级进行预处理并乘以权重，加和之后输出预测星级。

下面我将对每一个步骤进行详细介绍。

B. 数据预处理及特征统计

对一段评论来说，将其中的关键词语选出是非常关键的一步。我们使用了中文分词中常用的 jieba（中文名：结巴）库，该库具有分词、词性统计、语义权重统计等功能，能够较好地满足我们团队数据预处理的需求。由于 jieba 库会将所有词语进行不加选择的区分，而一段评论中某些词性的词语是对星级评定无关紧要的，所以在对评论进行分词时，我们对入库词语进行了筛选。

以下词性是我们在分词中所需要的。

TABLE I
在分词中保留的词性

中文名称	对应标号
形容词	a
副形词	ad
动词	v
副动词	vd
名动词	vn
副动词	z
成语	i
其他专名	nz
名词	n
习用语	l

词语从一段话中提取出来之后，我们将这段评论的星级赋给评论中所有分出来的词语，如下是一个例子。

TABLE II
将词语赋予该评论的对应星级

原始评论	星级	分词后赋予词语星级
第一部真的很好看，最喜欢的超级英雄，无可比拟。	5	无可比拟 5 星 好看 5 星 喜欢 5 星

随后，我们对数据集中的所有评论数据都进行如上操作，将出现多次的词语星级取历史星级平均值，即可得到词语在该数据集中的平均分数，将保存各词语平均星级、出现总次数的文件保存为 word_dict.csv 文件，即可完成数据预处理和特征统计工作。

C. 测试数据分词及参数预处理

在对整个训练样本进行词频统计和单词平均星级统计后，将测试样本（评论数据）进行分词即可得到测试样本的特征词语。利用 jieba 库内置函数可以得到各特征词语在该评论中的权重（即重要程度），一个词语越重要，那么这个词的权重越高。但是值得注意的是，jieba 库输出的权重并不是归一化的，且当评论较长、关键词较多时，权重差别并不大。因此我在程序中对 jieba 库输出的词语权重进行了 Laplace smoothing 平滑处理，具体信息可以参照代码。处理之后的词语权重即能更好体现一段评论中词语的重要程度。

另外通过对数据集的分析发现，一个词语所表达的“褒贬程度”，不仅与平均星级有关，还与该词语在总训练样本中出现的次数有关。具体来讲，一个词语出现的次数越多，那么这个词语就会越倾向于中位数 3 分。因此，在将词语的平均星级代入权重相乘时，我们首先利用函数计算将实际星级变为原始平均星级和出现次数的二元函数，以此来获得更加准确的星级评价。

在分别对测试样本中的词语权重和星级进行预处理之后，我们即将该词语的权重与其星级相乘并求和，即可算出整个评论的预测星级。

D. 算法评估

以上方法中，我们使用了含有约 32000 条评论的训练样本，制作了一个包含所有词语平均星级和出现次数的字典数据集。我们另外选取了含有 4500 条评论的测试数据集，在此测试数据集上测试上述方法的精度。

在测试中，我们采用了三种精度评判方法。第一种方法是测试星级四舍五入与实际星级严格相等时判定为预测正确时的精度。第二种方法是测试星级落在实际星级两侧时判定为预测正确时的精度（如测试星级为 2.518，那么实际星级为 2 星和 3 星都算作判定正确）。第三种方法是测试星级四舍五入落在实际星级两侧时判定为预测正确时的精度（如测试星级为 3.168，四舍五入为 3，则判定为 2、3、4 星均为正确）。

将测试样本代入网络中，统计测试星级和实际星级的匹配程度。当采用第一种评判标准时，精度约为 36.36%；当采用第二种评判标准时，精度约为 66.16%；当采用第三种评判标准时，精度约为 83.12%。

一般而言，第二种评判标准可能更为科学，因为其具有一定的灵活性，能够反映电影评论的主观因素。

E. 方法缺陷及改进

以上算法还存在诸多缺陷，具体描述如下。

首先是算法逻辑性和学习能力还有待加强。总的来说，我们采用基于统计的贝叶斯方法，并根据问题的实际情况进行了修改。但是基于概率模型的方法其实不能很好地体现一段评论中的语境和语义，只能全部依赖其中的词语进行判断。而由于词语的多义性，我们分类的正确率会较快遇到瓶颈。为解决此方法，我们稍后会采用基于神经网络的算法进行实现。

另外，jieba 分词的质量还有待提高。我们采用了多种词语筛选方法防止无用词语出现在库中，但是效果不尽人意，这也直接影响了我们后期的测试精度。

IV. 基于词嵌入的简单神经网络

A. 词嵌入背景介绍

词嵌入是自然语言处理中常用的数据降维方式，早期实现是基于词的分布假说（上下文的词之间有语义上的联系），并使用无监督学习的方法，将每一个词用一个低维的向量表示出来，从而达到数据降维的目的。现在的词嵌入方法一般是基于神经网络的 [1]。

2013 年，谷歌公司开发了 word2vec 工具，这是一个基于神经网络的词向量训练包。它可以根据给定的语料库，通过优化后的训练模型快速有效地将一个词语表达成向量形式，为自然语言处理领域的应用研究提供了新的工具。word2vec 依赖 skip-grams 或连续词袋（CBOW）来建立神经词嵌入。

虽然 word2vec 在训练词向量方面有着很强的能力，但是无法与随后我们自己构建的神经网络产生连接。也就是说，word2vec 产生的词向量是脱离任务产生，只是单纯生成模型，给出一种词的表达方式。如果这种表达方式没有包含词的情感倾向信息，或者说表达的不够明显，那么无论我们如何训练后面的神经网络，都不可能得到正确的答案。

根据“《How to Generate a Good Word Embedding?》导读”这篇文章的研究结果，我们了解到词向量表示结果的好坏和语料库的大小以及领域有关。落实到具体问题上，我们所要研究的是豆瓣短评的情感倾向问题，而一般中文的语料库会选取《人民日报》的语料来进行训练（因为比较好找而且用语规范），这两者之间是有较大区别的，豆瓣短评中网络用语、口语化表达比较常见，而在《人民日报》语料中，几乎不会出现网络用语和过于口语的表达。这样不利于后面的模型提取到短评中的关键信息，从而影响判断。

抛弃了构建语言模型和训练分类神经网络相分离的想法之后，我们决定使用 tensorflow 中现有的 embedding 层。embedding 层使用投入的语料来学习获得词向量，这样会使我们的词向量更加符合我们的应用环境。除此之外，embedding 也可以和后面的网络结构相连接，方便了我们的训练和测试。

B. 数据预处理

为更好地完成此任务，我们的数据集经过了三代的变化。第一次我们只爬取了 21 部漫威电影一共 3 万多条的评论，并没有单独提取出测试集，测试过程都是使用 tensorflow 自带的 split 功能，随机将训练集的一部分（约 10%）作为验证集。第二次，我们爬取了三部 DC 超级英雄电影共 4500 条短评作为验证集。以上共称为第一版数据集。第三次扩充则为动作科幻片合集，一共约 20 万条初步数据，称为第二版数据集。

在查看所爬得的数据后，我们发现有部分短评没有评分，只有标点或英文，以及大量表情符号。于是我们在删除了一部分不符合要求的短评之后，去掉剩余评论中所有的英文、标点、表情符号，只保留中文，投入 jieba 分词中。将所有出现的词做成一个列表，第一个数据集共有四万七千多词，在扩充了数据集之后，第二个数据集将近有 12 万词。将数据集按照词表的顺序标号，即每一条短评在分词之后都会变成一个一维的整数数组。为了方便，我们将每一条短评都用 0 补足到 256 维，如果有一些词库中未曾出现的词（在验证集中出现而训练集中未出现过的情况）则用 0 代替。这样就得到了可以投入模型的数据集文件。

C. 模型构建以及训练

具体模型的构建参看 Fig.1 与 Fig.2 的内容

模型构建上，由于选取了较低维度的词向量，所以隐层神经元个数只有 16 个，最后是 5 个神经元的 softmax 输出层用来得到预测星级的概率。明显可以看出，第二版数据集的参数量要比第一版数据集的参数量大很多，因为词库扩大了一倍多。但是由于 embedding 采取的是负采样的方法，所以两个模型都训练的比较快。第一版数据集每轮只需要 6s，第二版数据集每轮也只需要 50s 至 60s。

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, None, 32)          3812064
_________________________________________________________________
global_average_pooling1d_4 ( (None, 32)                0
_________________________________________________________________
dense_8 (Dense)              (None, 16)                528
_________________________________________________________________
dense_9 (Dense)              (None, 5)                 85
=================================================================
Total params: 3,812,677
Trainable params: 3,812,677
Non-trainable params: 0
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 64)          7624128
_________________________________________________________________
global_average_pooling1d (Gl (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 16)                1040
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 85
=================================================================
Total params: 7,625,253
Trainable params: 7,625,253
Non-trainable params: 0
_________________________________________________________________
```

Fig. 1. 基于第一版数据集的网络构建，上图为词向量 32 维的情况，下图为词向量 64 维的情况

D. 结果比较和分析

模型结果如 Fig.3 所示，图例中的数字 1、2、3、4 分别代表第一版数据集-32 维词向量、第一版数据集-64 维词向量、第二版数据集-32 维词向量、第二版数据集-64 维词向量。验证集均为包含 4500 条短评的 DC 电影评论验证集。以上训练方式完全相同，均采用 adam 优化器，以默认方式训练。

训练结果如下（第 100 轮训练的最终结果）

- 第一版数据集-32 维词向量
  acc:84.63% val_acc:39.02%
- 第一版数据集-64 维词向量
  acc:84.86% val_acc:39.07%
- 第二版数据集-32 维词向量
  acc:74.70% val_acc:27.51%
- 第二版数据集-64 维词向量
  acc:73.57% val_acc:29.56%

在训练过程中，四个模型都出现了过拟合的现象，训练集的准确率逐渐上升，但是测试集的精度却有小幅度的

Layer (type)	Output Shape	Param #
embedding_4 (Embedding)	(None, None, 32)	3812064
global_average_pooling1d_4 (	(None, 32)	0
dense_8 (Dense)	(None, 16)	528
dense_9 (Dense)	(None, 5)	85

Total params: 3,812,677
Trainable params: 3,812,677
Non-trainable params: 0

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, None, 64)	7624128
global_average_pooling1d (Gl	(None, 64)	0
dense (Dense)	(None, 16)	1040
dense_1 (Dense)	(None, 5)	85

Total params: 7,625,253
Trainable params: 7,625,253
Non-trainable params: 0

Fig. 2. 基于第二版数据集的网络构建，上图为词向量 32 维的情况，下图为词向量 64 维的情况

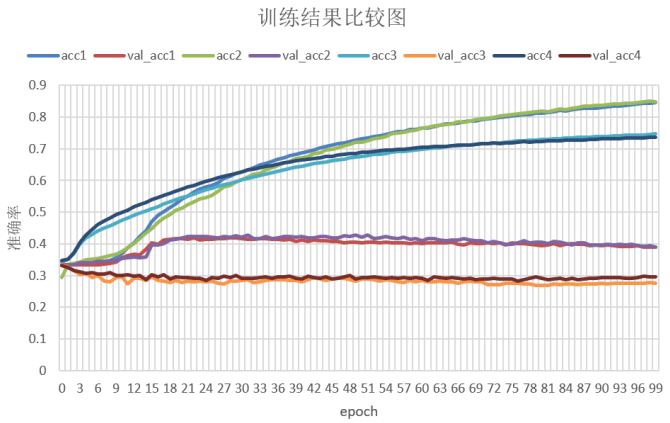

Fig. 3. 不同参数训练结果比较图

下降。不过我们发现，小数据集上的效果要比大数据的效果好很多。但这很大程度上是因为，验证集和小数据的契合度较高，因为类型和题材很接近，都是超级英雄类型电影，有动作和科幻的元素，观影群体也比较相似。而大数据集里除掉小数据集已经包含的内容之外，还有大量的动作科幻电影，这些电影的内容种类多且繁杂，观影人群广泛，虽然泛化性能可能会更好，但是在只有超级英雄电影这一种类型的验证集上表现难免差强人意。

另外，从训练集精度上来说，大数据集也比不上小数据集。这很大程度是因为大数据集内部的矛盾性要大于小数据集内部的矛盾性，没办法训练得到一个契合数据的模型。可以观察到，小数据的训练集精度上升要快于大数据集的训练集精度。而且小数据集的验证集精度可以保持在 40% 左右，而大数据集基本上只能维持在 30% 左右。

出于应用的目的，我们觉得不能只评估模型的泛化能力，也就是验证集精度，更需要综合考虑训练集精度。所以，我们采用训练集精度乘测试集精度来表示模型的表达能力。具体表示如下。

$$EA = \text{train\_acc} * \text{val\_acc} * 100$$

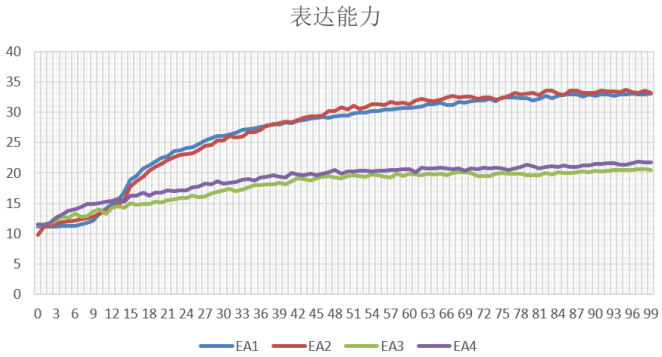

Fig. 4. 不同参数模型的表达能力

按照我们的标准，最终表达能力较好的应该是小数据集的两个模型。从结果上我们也能看出，词向量的选取 32 维还是 64 维对结果没有特别大的影响。

V. LSTM 网络

A. RNN 与 LSTM 介绍

由于本项目属于自然语言处理 (NLP) 方面的工作，所以我们查看了大量有关影评情感分析方面的教程。发现由于 NLP 问题的处理中涉及到前后文 (词语) 的关联性即存在顺序的信息，所以主流的用于处理 NLP 问题的网络是能够处理序列数据的 RNN 网络。而 LSTM（全称为 Long short-term memory）是一种特定形式的 RNN [2]。

一般地，RNN 包含如下三个特性：

- 循环神经网络能够在每个时间节点产生一个输出，且隐单元间的连接是循环的
- 循环神经网络能够在每个时间节点产生一个输出，且该时间节点上的输出仅与下一时间节点的隐单元有循环连接
- 循环神经网络包含带有循环连接的隐单元，且能够处理序列数据并输出单一的预测。

RNN 还有许多变形，例如双向 RNN（Bidirectional RNN）等。然而，RNN 在处理长期依赖（时间序列上距离较远的节点）时会遇到巨大的困难，因为计算距离较远的节点之间的联系时会涉及雅可比矩阵的多次相乘，这会带来梯度消失（经常发生）或者梯度膨胀（较少发生）的问题，这样的现象被许多学者观察到并独立研究。为了解决该问题，研究人员提出了许多解决办法，例如 ESN（Echo State Network），增加有漏单元（Leaky Units）等等。其中最成功应用最广泛的就是门限 RNN（Gated RNN），而 LSTM 就是门限 RNN 中最著名的一种。有漏单元通过设计连接间的权重系数，从而允许 RNN 累积距离较远节点间的长期联系；而门限 RNN 则泛化了这样的思想，允许在不同时刻改变该系数，且允许网络忘记当前已经累积的信息。

LSTM 就是这样的门限 RNN，LSTM 的巧妙之处在于通过增加输入门限，遗忘门限和输出门限，使得自循环的权重是变化的，这样一来在模型参数固定的情况下，不同时刻的积分尺度可以动态改变，从而避免了梯度消失或者梯度膨胀的问题。

B. 从初步短评资料库到网络的输入

在获取了短评与评分的 csv 文件后，我们需要考虑的是如何将一段影评转化成 LSTM 网络的输入。数据预处理的基本方法与前两种方法使用的预处理类似，都是将一段话通过某种方法转换为网络可以接受的输入形式。

由于影评中存在许多诸如表情符，标点符号，特殊符号等许多与情感分析无关的内容以及文字中许多和情感无关的词汇例如地点词，人名，虚词等。这些干扰因素都需要去除，所以我们发现并借助了结巴中文分词库中关键词提取的功能。

利用结巴分词中的 analyse.extract_tags 功能，我们将每一个短评中前 40 个重要的关键词提取出来（不足 40 个的就按其能提取出来的最多的关键词数量），为了利用到关键词的顺序信息，我们将关键词按照其在短评中出现的顺序进行排列，得到初步的数据集 new_data.csv。

接下来是如何将数据集 new_data.csv 中包含的顺序关键词转化成网络的输入。很明显的是，一个网络的输入只能是数字而不能是文字。在查阅了相关资料后，我们决定先利用得到的所有的关键词做成一个词语语料库，每个关键词词语在语料库中都有一个相应的编号（即序号），接着遍历每一个顺序关键词 list，根据每个关键词在语料库中的编号即可将顺序关键词 list 转化成对应的数字 list，例如第一条短评 [' 这是', ' 一个', ' 搞笑', ' 臭屁', ' 喋喋不休', ' 英雄', ' 结尾', ' 那句', 'am', 'Iron', 'Man', ' 太帅'] 转化成 [1,2,3,4,5,6,7,8,9,10,11,12]。保留数字 0 用于表示未在词典中出现的词以及补位使用。最终得到可以投入网络的数据集 features.npy 和 labels.npy。

C. 构建 LSTM 网络

在得到可投入网络的数据集 features.npy 和 labels.npy 后，我们开始着手以构建 LSTM 网络。

网络的主要有三块组成：Embedding 层，LSTM 层和全连接层 [3]。

由于 LSTM 层的输入形式为一个句子矩阵 (此处为顺序关键词矩阵)，所以我们需要在 LSTM 层前加入一个 Embedding 层。Embedding 层的作用是将一个大小为 (1*m) 的短评顺序关键词 list 转化成一个大小为 (m*n) 的短评顺序关键词矩阵，其中 m 代表关键词的个数，n 代表用于表示一个关键词的词向量维数。n 可以自由设定，且网络对句子长度 (此处为关键词个数) 不敏感。

LSTM 层的输入句子矩阵，输出可选择为所有单元输出即句子矩阵（多层 LSTM 相接）或最后一个单元的输出（后接全连接层），重要的可调参数为 units 即输出的维度。

全连接层：由多层 Dense 组成，最后一层输出为 5 即可。

D. 网络训练效果与调整尝试

最初我们只爬取了 20 部漫威系列电影共 3 万条短评与评分信息作为训练集，同时爬取了 3 部 DC 科幻电影电影作为测试集，利用 keras 中的 EarlyStopping 模块设置测试集的预测集精度为早停指标以及 patience 为 10 即测试精度下降后继续训练 10 个 epochs，网络构成和训练结果如 Fig.5 与 Fig.6 所示：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 40, 20)            978640
_________________________________________________________________
lstm (LSTM)                  (None, 80)                32320
_________________________________________________________________
dense (Dense)                (None, 20)                1620
_________________________________________________________________
activation (Activation)      (None, 20)                0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 105
_________________________________________________________________
activation_1 (Activation)    (None, 5)                 0
=================================================================
Total params: 1,012,685
Trainable params: 1,012,685
Non-trainable params: 0
```

Fig. 5. 网络组成

```
Train on 31955 samples, validate on 4500 samples
Epoch 1/50
31955/31955 [==============================] - 34s 1ms/step - loss: 1.4556 - acc: 0.3351 - val_loss: 1.4930 - val_acc: 0.3331
Epoch 2/50
31955/31955 [==============================] - 29s 920us/step - loss: 1.4375 - acc: 0.3365 - val_loss: 1.4887 - val_acc: 0.3384
Epoch 3/50
31955/31955 [==============================] - 29s 898us/step - loss: 1.4079 - acc: 0.3406 - val_loss: 1.4731 - val_acc: 0.3236
Epoch 4/50
31955/31955 [==============================] - 28s 883us/step - loss: 1.3811 - acc: 0.3614 - val_loss: 1.4502 - val_acc: 0.3482
Epoch 5/50
31955/31955 [==============================] - 28s 861us/step - loss: 1.3296 - acc: 0.4033 - val_loss: 1.4027 - val_acc: 0.3771
Epoch 6/50
31955/31955 [==============================] - 27s 857us/step - loss: 1.2480 - acc: 0.4578 - val_loss: 1.4119 - val_acc: 0.3884
Epoch 7/50
31955/31955 [==============================] - 27s 843us/step - loss: 1.1539 - acc: 0.5109 - val_loss: 1.4342 - val_acc: 0.3802
Epoch 8/50
31955/31955 [==============================] - 27s 848us/step - loss: 1.0551 - acc: 0.5669 - val_loss: 1.5299 - val_acc: 0.3842
Epoch 9/50
31955/31955 [==============================] - 27s 833us/step - loss: 0.9633 - acc: 0.6133 - val_loss: 1.6676 - val_acc: 0.3844
Epoch 10/50
31955/31955 [==============================] - 27s 839us/step - loss: 0.8793 - acc: 0.6516 - val_loss: 1.7083 - val_acc: 0.3853
Epoch 11/50
31955/31955 [==============================] - 27s 854us/step - loss: 0.7982 - acc: 0.6885 - val_loss: 1.8331 - val_acc: 0.3818
Epoch 12/50
31955/31955 [==============================] - 29s 907us/step - loss: 0.7266 - acc: 0.7243 - val_loss: 1.9672 - val_acc: 0.3829
Epoch 13/50
31955/31955 [==============================] - 28s 872us/step - loss: 0.6589 - acc: 0.7482 - val_loss: 2.1055 - val_acc: 0.3749
Epoch 14/50
31955/31955 [==============================] - 26s 827us/step - loss: 0.6000 - acc: 0.7681 - val_loss: 2.2244 - val_acc: 0.3724
Epoch 15/50
31955/31955 [==============================] - 27s 840us/step - loss: 0.5514 - acc: 0.7871 - val_loss: 2.2725 - val_acc: 0.3587
Epoch 16/50
31955/31955 [==============================] - 27s 832us/step - loss: 0.5109 - acc: 0.8068 - val_loss: 2.3787 - val_acc: 0.3667
```

Fig. 6. 训练过程及结果

从训练结果可以看到，在训练过程中训练集的精度一直在提高，但训练集的精度提高的非常有限最高只有 38.84% 并且之后测试集的精度降低导致训练过程早停，此时训练集的精度已经达到了 80.68%，很明显发生了过拟合现象。

经过讨论，我们认为其实精确的需要预测结果和评分完全一致是没有必要的。例如一个人认为一部电影不好看，他可能给出 1 星，也可能给出 2 星，给定的星级与个人主管因素有很大的关系，所以接下来我们决定，将 1、2 星合为差评，3 星记为中评，4、5 星记为好评即输出结果仅为 3 类。网络结构中仅将最后一层的 Dense 输出改为 3，结果如 Fig.7 所示：

从训练结果可以看到，将影评的标签分为 3 类后，随着训练的进行，训练集的精度不断上升，测试集的精度先上升到最大值 50.53% 后逐渐下降并导致提前终止训练。可见，即使降低了一些预测难度，目前的网络模型表现仍不太好，在验证集上仅有 50 左右的正确率。

经过讨论，我们认为这样简单的分为 3 类的要求还是有点高，因为个人的主观评分因素影响还是较大，例如一个影评一个人打的是 2 星，那么其实了另一个人做出相似的影评后，打 1 星、2 星或 3 星都是可以理解的，即在预测

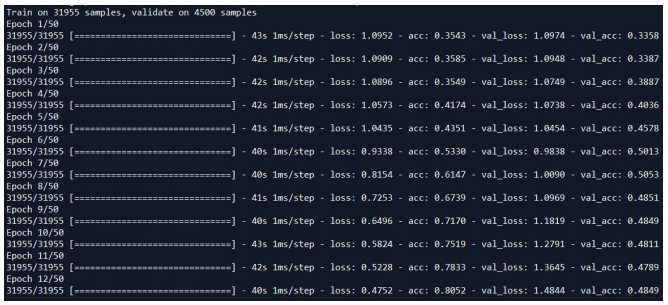

Fig. 7. 评判标准调整之后的训练过程及结果

的过程中，对于一个实际标签为 3 星的影评，那么预测结果为 2 星、3 星、4 星我们都认为其预测准确。将预测精度的计算方式更改了之后再次进行模型泛化能力的测试，即将不在训练集中的影评放入进行预测，结果如 Fig.8 所示：

```
In [7]: test = pd.read_csv('test.csv')
   ...: comment = test['comment']
   ...: score = test['score']
   ...: count = 0
   ...: for i in range(len(comment)):
   ...:     number_input = get_test_number_list(comment[i])
   ...:     predict = np.argmax(model.predict(number_input)) + 1
   ...:     if (int(score[i])-1) <= predict and predict <= (int(score[i])+1):
   ...:         count += 1
   ...:
   ...: print('test_acc: ' , count / len(comment))
test_acc:  0.6693333333333333
```

Fig. 8. 再次调整评判标准之后的训练过程及结果

由实验结果看，再次放低了了准确范围的要求后，测试集的精度约为 67%，比第二次（三种分类）的结果好了一些。

由于我们认为 3 万的数据集偏小，并且由于我们的测试集选择的是未在训练集中出现的词，在词语字典中未出现的词而补零的情况会较大，所以我们决定增大数据集，另外爬取了约 100 部科幻/动作电影共 19 万的影评及其评分信息（即在词向量部分提到的第二版数据集）。在经过与之前相同的预处理后，投入网络进行训练，训练结果如 Fig.9 所示：

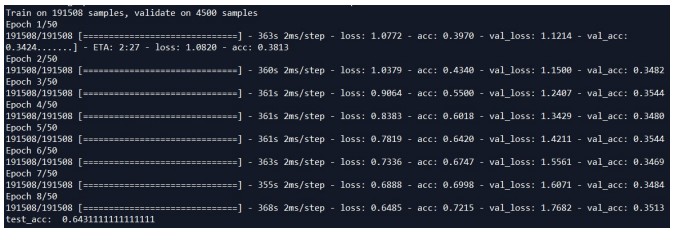

Fig. 9. 扩增数据集之后的训练过程及结果

从训练结果可见，虽然增大了数据集，但网络的表现并没有变好，训练集的精度随着训练次数不断增大，而测试集的精度几乎没有增大，最终测试集的精度还比 3 万数据集的差一些，经分析，应该是由于词库成倍扩大后类似的词向量样本稀疏导致的。

经过多次的尝试，我们的 LSTM 网络预测结果并没有达到预期中非常高的精确度，认为有两方面的原因。

首先是数据的预处理没做好。拿到影评和评分的数据后若直接用结巴精确分词后投入网络训练，干扰的无效信息太多，可能需要巨大量的数据或者需要十分的复杂网络以及非常久的训练时间。若我们先用结巴分词里的关键词提取功能，虽然按照各个关键词在句中出现的次序排列，但终究不是完整的句子，还是丢失部分序列信息并且只提取关键词前后连贯性也不好。

其次是对 LSTM 的了解不够深刻，利用程度不足。由于我们没有系统的学习过 RNN 网络，在网络上查找相应的教程后虽然前前后后看了大约一周时间，最终实现的时候还是选择用 keras 中的 Embedding 层加 LSTM 层这样集成度很高的方式。而情感分析主要是一些关键的表示情感的词语以及表示加强或否定的词语在起作用，推测 LSTM 用于用户输入一些文字后预测将要使用的词语的效果会比较好。在学习 LSTM 的过程中发现 NLP 中很多问题的实现都是用的英文词库，并且输入对象是整个句子而非分词后的关键词序列，经过 word2vector 库可以将每个单词转化成词向量，之后的思路和中文处理一致。很明显，在经过结巴分词后丢失了部分的序列信息是很大一部分预测效果不理想的原因。相比中文的博大精深，对英文处理可以减少许多不必要的麻烦。如果时间允许，后期我们还有一个思路是先将影评经过谷歌翻译得到相应的英文评论，再利用英文评论进行网络的训练，当输入一个中文影评时，也是先经过翻译后得到相应的英文评论，经过 word2vector 库的处理后，再输入已训练好的网络进行预测。

VI. 结果可视化

A. 词云

根据我们得到的关键词库，我们统计了每个关键词在所有影评中出现的次数及其平均分，从而得到相应的关键词-平均分-出现频次的 word_dict.csv 文件。

经过观察，平均分在 2.95-3.01 区间的大部分关键词与评分好坏无关，所以在显示时舍去了这一部分的词。利用 Wordcloud 库中的 generate_from_frequencies 功能，根据关键词及其出现频次绘制出影评中关键词词云如 Fig.10 所示：

Fig. 10. 根据词语的出现频率构建可视化词云

B. 模型交互界面设计

为使用户有比较完美的交互体验，我们设计了一个简单的交互窗口供用户使用。用户可以在交互窗口中输入自

己的评论，随后评论被发送到后台的网络模型中（可以是我们尝试过的三种模型中的任意一个）。交互界面如 Fig.11 所示：

Fig. 11. 交互界面

在交互界面的文本框中输入影评点击预测即可在粉色方框中得到预测的评分，点击清空即可重新输入评论并预测星级。

VII. 项目总结

在这四周的时间里，我们通过组内良好的合作及分工，较好地按照最初的项目计划时间节点，不断推进项目进程，最终得到了针对科幻和动作电影的影评评分预测系统。

通过这次项目，我们在实践中亲身经历了数据的获取，数据的预处理，分类方法的选取，网络的设计与调整，最终结果的评价与可视化一系列项目操作过程，对处理 NLP 问题的具体流程也有了一定的了解。

期间，我们为达到项目要求的过程中不断学习和尝试新方法，利用的方法有：post 模拟登陆进行爬虫，jieba 库进行影评预处理，word2vector 库进行词向量的转化，基于词性质的朴素贝叶斯分类法，基于词嵌入的简单神经网络，基于词嵌入的 LSTM 网络以及最终的词云绘制和形成交互界面。

最后，我们要感谢屠老师和助教学长，你们在我们遇到问题时给与了悉心的指导和帮助，使我们有了思考的方向和信心，也感谢同组队友间的相互体谅和支持，最终得以共同完成了这个项目。

References

[1] Tzu-Ray Su and Hung-Yi Lee, Learning Chinese Word Representations From Glyphs Of Characters, 2017.
[2] Shalini Ghosh, Oriol Vinyals, Brian Strope, Scott Roy, Tom Dean, Larry Heck, Contextual LSTM (CLSTM) models for Large scale NLP tasks (Submitted on 19 Feb 2016 (v1), last revised 31 May 2016 (this version, v2))
[3] Zhiheng Huang, Wei Xu, Kai Yu, Bidirectional LSTM-CRF Models for Sequence Tagging, 2015