

上海對外經貿大學

## 《自然语言处理》课程报告

报告题目: IMDB 电影评论情感分析

小组成员: 19024075 陶盛皿

完成日期: 2022 年 05 月 08 日

## 一、研究背景及主要研究内容

本报告主题为：IMDB 电影评论情感分析，开源数据集来自 Kaggle。

本报告运用自然语言处理技术（包括：英文分词、基于统计的方法 TF-IDF、词向量），使用的训练模型，包括：

- （1）基于统计方法 TF-IDF 和词袋向量使用 Logistic Regression 模型；
- （2）基于 Word2Vec 的方法使用随机森林模型、Logistic Regression 模型、XGBoost 模型，并使用 GridSearchCV 调参；
- （3）基于深度学习的方法，结合 Bert 预训练的 Transformer 模型和使用 Bert 预训练的 Bert 模型）

对数据集的电影评论进行文本分类，预测评论的积极性或者消极性。最后，根据准确率对比实验结果。

## 二、数据集介绍

该数据集具有三个文件：训练集、测试集和未标注集。训练集数据，还有三个变量，其中 id 是样本唯一索引标识符，review 是电影评论内容，sentiment 为情感得分（“0”为消极评论，“1”为积极评论）。

	id	sentiment	review
0	5814_8	1	With all this stuff going down at the moment w...
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi...
2	7759_3	0	The film starts with a manager (Nicholas Bell)...
3	3630_4	0	It must be assumed that those who praised this...
4	9495_8	1	Superbly trashy and wondrously unpretentious 8...

图表 1 训练集

### 三、 本实验的方法汇总

#### a) 基于统计的方法

##### i. TF-IDF 和词袋向量

TF-IDF 是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。TF-IDF 的主要思想是：通过统计词频来反映词的权重高低。如果词频高，则该词能够很好代表这个类的文本的特征，故被赋予较高的权重。本方法直接使用 `sklearn.feature_extraction.text.TfidfVectorizer` 将原始文本转化为 TF-IDF 特征矩阵，统计词频计算文本相似度。其中参数 `sublinear_tf = True`，即将 TF 替换为  $1 + \log(TF)$ ，参数 `analyzer = 'word'`，即一个词元 (token) 为一个特征，参数 `stop_words = 'english'`，即直接过滤停用词，参数 `gram_range = (1, 1)`，即使用双字符 (2-gram)，从训练集语料中学习词汇和 IDF，并将其转换为词元特征矩阵。同时，字符向量额外添加参数 `strip_accents = 'unicode'`，即对字符标准化，调整参数 `gram_range = (2, 6)`，得到字符特征矩阵。

##### ii. 模型训练

切分数据集，使用 Logistic 回归模型，损失函数使用随机梯度下降。用控制变量的思想运用 GridSearchCV 逐一调参，使模型准确率达到最佳。

#### b) 基于机器学习的方法

##### i. 数据预处理：数据清洗和 NLTK 分词

首先，使用正则表达式对评论文本进行数据清洗，即：去除网页标志符、去除 emoji 表情符号、去除音调符号等。其次，进行字符串标准化。最后，使用 NLTK 进行分词，建立语料库，并过滤长度非法的词和停用词。

##### ii. 特征工程：使用 word2vec 进行 Embedding

词的向量化就是将自然语言中的词语映射成是一个实数向量，用于对自然语言建模，比如进行情感分析、语义分析等自然语言处理任务。本方法未使用基于

词袋 (bag of words) 的编码方法，直接使用 word2vec。

word2vec 是以无监督方式从文本语料中学习富含语义信息的词向量的语言模型，将词 (token) 从原先所属的空间映射到新的低维空间，因为词向量具有很好的语义特性，并加入了词的顺序信息。

### iii. 模型训练与调参

本报告使用三种使用频率高的分类学习模型：随机森林、Logistic 回归、XGBoost 模型，用控制变量的思想运用 GridSearchCV 逐一调参，使模型准确率达到最佳。

## c) 基于深度学习的方法

### i. Bert 预训练模型 + Transformer

#### 1. 分词与编码

本方法与机器学习方法相同之处是依旧采取正则表达式和 NLTK 去除进行数据清洗（去除网页标识符、去除 emoji 表情、去除音标编码、字符标准化）。导入 transformers 包，并使用 bert-based-uncased 预训练模型的分词函数 `tokenizer = transformers.AutoTokenizer.from_pretrained('bert-based-uncased')`，`tokenizer.tokenize` 进行直接分词，使用 `tokenizer.encode` 进行编码。（注：tokenizer 也可以进行反向编码以及索引 PAD 的编码以及词元。）

#### 2. Transformer 架构导入和模型创建

直接调用 `transformers.AutoModel.from_pretrained('bert-based-uncased')` 即可创建 Transformer 架构（Encoder-Decoder 架构），总体模型建立依旧采用面向对象 `class Transformer(nn.Module)` 形式包装，加入全连接层和前向传播函数，详细代码见附录。

#### 3. GPU 和模型训练

(1) GPU: Tesla T4 NVIDIA-SIM 460.32.03, CUDA Version: 11.2

(2) 学习率:  $1e-5$

(3) 优化器: `optim.Adam(model.parameters(), lr = 1e-5)`

(4) 交叉熵损失函数: `nn.CrossEntropyLoss()`

(5) 使用批量训练法方式进行训练和验证。

## ii. Bert 模型

### 1. 分词和编码

本方法与 Transformer 方法相同之处是依旧采取正则表达式和 NLTK 去除进行数据清洗（去除网页标识符、去除 emoji 表情、去除音标编码、字符标准化）。在文本清洗之后，导入 `transformers` 包，并使用 `bert-based-uncased` 预训练模型的 Bert 分词函数 `tokenizer = transformers. BertTokenizer.from_pretrained('bert-based-uncased')`, `tokenizer.tokenize` 进行直接分词, 使用 `tokenizer.encode` 进行编码。(注: `tokenizer` 也可以进行反向编码以及索引 PAD 的编码以及词元。)

### 2. Padding 和创建注意力 mask

导入 `keras.preprocessing.sequence.pad_sequences`, 设置最大长度 `MAX_LEN = 128` 进行 Padding, 目的是区别 Transformer 方法中的批量训练方式, 使用层训练(即相当于把序列当做一个批量)。`pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long", value=0, truncating="post", padding="post")` 获取输入序列的编码, 并使用 0 在非法位置占位进行 Padding 操作。

在获得 padding 之后, 对输入序列编码大于 0 的词元使用 1 来掩蔽, 若编码等于 0 的词元(即 “[PAD]”)使用 0 来遮蔽, 由此获得注意力 mask 矩阵。

### 3. 数据集切分

使用 `sklearn.model_selection.train_test_split` 将训练数据划分和训练集和验证集, 同时需要将注意力 mask 也划分训练 mask 和验证 mask, 并且转换成 Pytorch 的张量形式(避免报错, 使用 float 形式的 LongTensor)。

此处还需要使用 `torch.utils.data` 中的 `TensorDataset`, `DataLoader`, 将数据转换先转换成张量型数据集, 并且包装成迭代器形式。同时调用了 `torch.utils.data` 中

的随机采样 `RandomSampler` 和序列采样 `SequentialSampler` 进行数据采样和数据增强，加强模型捕获语义信息的能力。

#### 4. 模型创建

直接调用 `transformers.BertForSequenceClassification.from_pretrained` 并使用 'bert-based-uncased' 预训练模型，建立 Bert 序列分类模型。

#### 5. GPU 和模型训练

(1) GPU: Tesla P100 NVIDIA-SIM 470.80.01, CUDA Version: 11.4

(2) 学习率:  $2e-5$ , 丢弃率:  $1e-8$

(3) 优化器: `optim.Adam(model.parameters(), lr =  $2e-5$ , eps =  $1e-8$ )`

(4) 热身步: 使用 `transformers.get_linear_schedule_with_warm_up`, `num_warmup_steps = 100`。(注: 100 出自开源文档中 `run_glue.py` 以 GLUE 评估模型性能的经验值。)

(5) 使用序列和采样训练法方式进行训练和验证。

## 四、实验环境及实验结果

### a) 实验环境

方法	模型	平台	配置
统计	TF-IDF Logistic	Jupyter	本地 CPU
机器学习	NLTK + W2V RF/Logistic/XGB	Jupyter	本地 CPU
深度学习	Bert 预训练 Transformer 架构	Colab	Tesla T4 NVIDIA-SIM 460.32.03 CUDA Version: 11.2
深度学习	Bert 预训练 Bert 架构	Kaggle	Tesla P100 NVIDIA-SIM 470.80.01 CUDA Version: 11.4

## b) 实验结果

方法	模型	验证集准确率
统计	TF-IDF	最高 0.90000
	Logistic	
机器学习	NLTK + W2V	最高 0.87944
	RF/Logistic/XGB	
深度学习	Bert 预训练	最高 0.93800
	Transformer 架构	
深度学习	Bert 预训练 + 注意力 mask	最高 0.90000
	Bert 架构	

(详见附录截图。)

## 五、总结与展望

本报告融合了四种不同的实验（统计方法、机器学习方法、深度学习方法：Transformer 模型、深度学习方法：Bert 模型），虽然难度递增，但是深度学习方法在本次实验中只是提高了一定的准确率，并且增加了大量的参数开销。此外，本次实验过程中，使用 Bert 预训练的 Transformer 的准确率却高于完整 Bert 的模型，可能是因为偶然性因素，具体尚不清楚。

本次大作业过程中，收获颇丰：

- (1) 回顾了传统机器学习的方法；
- (2) 学会使用正则表达式进行文本预处理；
- (3) 读了 Transformer 论文，仔细研究了 Encoder-Decoder 架构以及注意力机制，并且了解了 Bert 模型对于 Transformer 的改进之处。
- (4) 学会在不同平台使用 GPU 和预训练模型。

不足之处和遇到的问题在于：使用深度学习模型，部署模型是可调节的参数是批量大小、Padding 的序列最大长度、学习率、训练周期和丢弃率，在模型架

构不变动的基础上，训练准确率提升较难。并且发现对于英文的自然语言处理相比中文比较简单，因为中文需要关注分词的粒度和歧义的问题，未来的展望：使用自然语言处理技术做中文分词和情感分析的实验。

## 六、附录

### a) 结果截图

```
classification_report
precision recall f1-score support
classification_report
precision recall f1-score support
0 0.81 0.84 0.82 2389
1 0.85 0.82 0.83 2611
accuracy 0.83 5000
macro avg 0.83 0.83 0.83 5000
weighted avg 0.83 0.83 0.83 5000
```

图表 5 W2V + Random Forest

```
classification_report
precision recall f1-score support
0 0.87 0.89 0.88 2442
1 0.89 0.88 0.88 2558
accuracy 0.88 5000
macro avg 0.88 0.88 0.88 5000
weighted avg 0.88 0.88 0.88 5000
Wall time: 3.37 s
```

图表 6 W2V + Logistic

```
classification_report
precision recall f1-score support
0 0.85 0.87 0.86 2418
1 0.88 0.86 0.87 2582
accuracy 0.87 5000
macro avg 0.87 0.87 0.87 5000
weighted avg 0.87 0.87 0.87 5000
```

图表 7 W2V +XGB



```

training...: 100%|██████████| 2344/2344 [16:39<00:00, 2.35it/s]
evaluating...: 100%|██████████| 782/782 [01:52<00:00, 6.96it/s]
epoch: 1
train_loss: 0.255, train_acc: 0.893
valid_loss: 0.191, valid_acc: 0.930
training...: 100%|██████████| 2344/2344 [16:41<00:00, 2.34it/s]
evaluating...: 100%|██████████| 782/782 [01:52<00:00, 6.94it/s]
epoch: 2
train_loss: 0.129, train_acc: 0.953
valid_loss: 0.173, valid_acc: 0.939
training...: 100%|██████████| 2344/2344 [16:40<00:00, 2.34it/s]
evaluating...: 100%|██████████| 782/782 [01:52<00:00, 6.94it/s]
epoch: 3
train_loss: 0.068, train_acc: 0.977
valid_loss: 0.181, valid_acc: 0.938

```

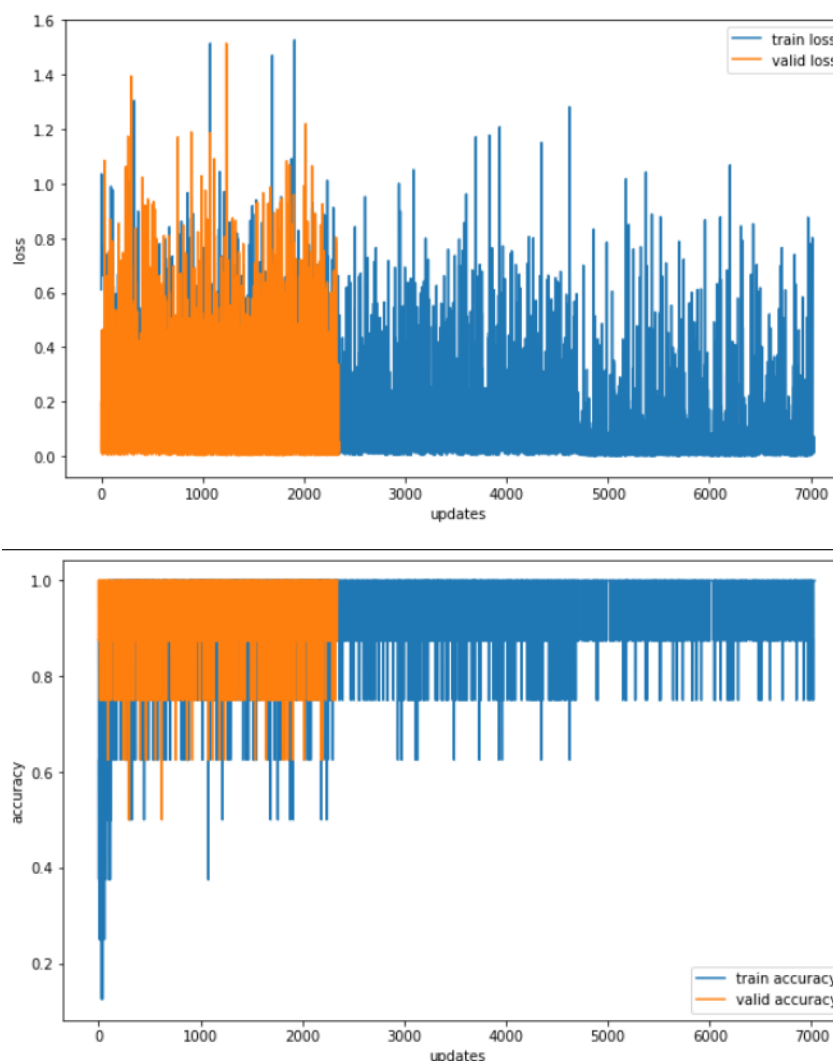
图表 8 Transformer Training

```

evaluating...: 100%|██████████| 3125/3125 [07:15<00:00, 7.18it/s]
test_loss: 0.170, test_acc: 0.938

```

图表 9 Transformer + Validation



图表 10 Transformer 损失图和准确率图

<pre> ===== Epoch 1 / 2 ===== Training... Batch 40 of 1,250. Elapsed: 0:00:09. Batch 80 of 1,250. Elapsed: 0:00:18. Batch 120 of 1,250. Elapsed: 0:00:27. Batch 160 of 1,250. Elapsed: 0:00:35. Batch 200 of 1,250. Elapsed: 0:00:44. Batch 240 of 1,250. Elapsed: 0:00:52. Batch 280 of 1,250. Elapsed: 0:01:01. Batch 320 of 1,250. Elapsed: 0:01:10. Batch 360 of 1,250. Elapsed: 0:01:18. Batch 400 of 1,250. Elapsed: 0:01:27. Batch 440 of 1,250. Elapsed: 0:01:35. Batch 480 of 1,250. Elapsed: 0:01:44. Batch 520 of 1,250. Elapsed: 0:01:53. Batch 560 of 1,250. Elapsed: 0:02:01. Batch 600 of 1,250. Elapsed: 0:02:10. Batch 640 of 1,250. Elapsed: 0:02:18. Batch 680 of 1,250. Elapsed: 0:02:27. Batch 720 of 1,250. Elapsed: 0:02:36. Batch 760 of 1,250. Elapsed: 0:02:44. Batch 800 of 1,250. Elapsed: 0:02:53. Batch 840 of 1,250. Elapsed: 0:03:01. Batch 880 of 1,250. Elapsed: 0:03:10. Batch 920 of 1,250. Elapsed: 0:03:19. Batch 960 of 1,250. Elapsed: 0:03:27. Batch 1,000 of 1,250. Elapsed: 0:03:36. Batch 1,040 of 1,250. Elapsed: 0:03:44. Batch 1,080 of 1,250. Elapsed: 0:03:53. Batch 1,120 of 1,250. Elapsed: 0:04:02. Batch 1,160 of 1,250. Elapsed: 0:04:10. Batch 1,200 of 1,250. Elapsed: 0:04:19. Batch 1,240 of 1,250. Elapsed: 0:04:27.  Average training loss: 0.35 Training epoch took: 0:04:30  Running Validation... Accuracy: 0.88 Validation took: 0:00:20 </pre>	<pre> ===== Epoch 2 / 2 ===== Training... Batch 40 of 1,250. Elapsed: 0:00:09. Batch 80 of 1,250. Elapsed: 0:00:17. Batch 120 of 1,250. Elapsed: 0:00:26. Batch 160 of 1,250. Elapsed: 0:00:34. Batch 200 of 1,250. Elapsed: 0:00:43. Batch 240 of 1,250. Elapsed: 0:00:52. Batch 280 of 1,250. Elapsed: 0:01:00. Batch 320 of 1,250. Elapsed: 0:01:09. Batch 360 of 1,250. Elapsed: 0:01:17. Batch 400 of 1,250. Elapsed: 0:01:26. Batch 440 of 1,250. Elapsed: 0:01:34. Batch 480 of 1,250. Elapsed: 0:01:43. Batch 520 of 1,250. Elapsed: 0:01:52. Batch 560 of 1,250. Elapsed: 0:02:00. Batch 600 of 1,250. Elapsed: 0:02:09. Batch 640 of 1,250. Elapsed: 0:02:17. Batch 680 of 1,250. Elapsed: 0:02:26. Batch 720 of 1,250. Elapsed: 0:02:35. Batch 760 of 1,250. Elapsed: 0:02:43. Batch 800 of 1,250. Elapsed: 0:02:52. Batch 840 of 1,250. Elapsed: 0:03:00. Batch 880 of 1,250. Elapsed: 0:03:09. Batch 920 of 1,250. Elapsed: 0:03:18. Batch 960 of 1,250. Elapsed: 0:03:26. Batch 1,000 of 1,250. Elapsed: 0:03:35. Batch 1,040 of 1,250. Elapsed: 0:03:43. Batch 1,080 of 1,250. Elapsed: 0:03:52. Batch 1,120 of 1,250. Elapsed: 0:04:00. Batch 1,160 of 1,250. Elapsed: 0:04:09. Batch 1,200 of 1,250. Elapsed: 0:04:18. Batch 1,240 of 1,250. Elapsed: 0:04:26.  Average training loss: 0.19 Training epoch took: 0:04:28  Running Validation... Accuracy: 0.89 Validation took: 0:00:20  Training complete! </pre>
---	---

图表 11 Bert 训练

```

Training epoch took: 0:00:47

Running Validation...
Accuracy: 0.90
Validation took: 0:00:19

Training complete!

```

图表 12 Bert 验证