上海对外经贸大学

《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）

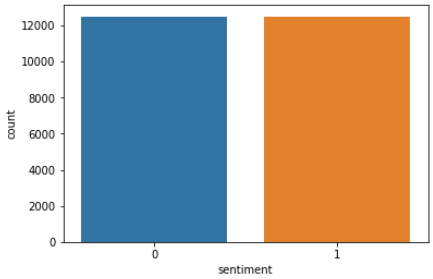
本实验对数据集的电影评论进行文本分类，预测评论的积极性或者消极性。最后，根据准确率对比实验结果，经过整个实验流程，相较于统计的方法，验证集准确率从最初 88%提升至 93.8%，总计提升 6.59%。

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。数据总量：50000，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）。该任务是需要通过建立分类模型，完成二分类预测问题，并且样本均衡。

	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...

```
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   id         50000 non-null  object
 1   sentiment  25000 non-null  float64
 2   review     50000 non-null  object
dtypes: float64(1), object(2)
memory usage: 1.1+ MB
```



图表 1 数据概览

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，用以评估一个字或者词对于一个文件集或一个语料库中的重要程度。TF-IDF 的主要思想是：通过统计词频来反映字或者词的权重高低，即如果词频高，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，反之，则其被赋予较低的权重。

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，统计词频并计算文本相似度。其中参数 sublinear_tf = True，即将 TF 替换为 1 + log (TF)的对数形式，参数 analyzer = 'word'，即将一个词元（token） 视为为一个特征（注：英文文本序列分词相对于中文文本序列分词较容易，英文文本序列中的词与词之间是直接使用空格连接，因此按照空格即可完成序列切分），参数 stop_words = 'english'，即直接过滤英文中停用词，参数 ngram_range = (1, 1)，即使用双字符（2-gram），从训练集预料中学习词汇和 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 ngram_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。

（由于在第二部分仍使用该模型，故此处不展开详细赘述）。

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。（代码详情见附录：关键代码截图。）其次，进行字符串标准化。最后，使用 NLTK 的分词函数进行英文分词，并过滤长度非

法的词，建立语料库。

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数值型的向量，用于对自然语言进行编码，随后输入模型进行模型训练，完成情感分析、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，而是直接使用 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，将词元（token）从原先所属的空间映射到新的低维空间，因此词向量具有很好的语义特性，并加入了序列中词元的顺序信息。

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，并使用控制变量的方法运用 GridSearchCV 逐一调参，目的是使用最优参数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

导入 transformers 包，并载入 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），总体模型微改需要建立面向对象定义 Transformer 类，即 class Transfomer(nn.Module)，在类中定义全连接层和前向传播函数，微改的方式是增加了一层全连接层，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) 学习率参数：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 方法不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，对非法位置进行 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. 学习率预热

学习率是神经网络训练中最重要的超参数之一，学习率预热是针对学习率的优化方式之一。在 ResNet 论文中提到的一种学习率预热的方法，它在训练开始的时候先选择使用一个较小的学习率，训练了一些轮数，再修改为预先设置的学习来进行训练。使用学习率预热的原因是：由于刚开始训练时，模型的权重是随

机初始化的，在预热的小学习率下，模型可以慢慢趋于稳定，等模型相对稳定后再选择预先设置的学习率进行训练，防止模型训练不稳定，使得模型收敛速度变得更快，提升总体算法效率。

4. 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 也划分层用于训练 mask 矩阵和用于验证 mask 矩阵，并且转换成 Pytorch 模型可接受的张量形式（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) 学习率参数：2e-5，丢弃率参数：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 Logistic	最高 0.90000
机器学习	NLTK + W2V RF/Logistic/XGB	最高 0.87944
深度学习	Bert 预训练 Transformer 架构	最高 0.93800
深度学习	Bert 预训练 Bert 架构	最高 0.90000

（详见附录截图。）

五、 总结与展望

a) 总结

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），难度递增，深度学习方法虽然提高了验证集的准确率，但是总体提升并不高。使用传统机器学习的分类方法相对于深度学习方法准确率并没有低太多，但是参数规模明显小得多。在这个数据集构造的语料库上，使用 Bert 预训练模型的准确率低于使用 Bert 预训练的 Transformer 模型可能是由于参数增多以及偶然性因素，具体原因不清楚。

本次大作业过程中，收获颇丰：

i. 回顾了传统机器学习的分类模型；

ii. 学会使用正则表达式进行文本预处理；

iii. 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；

iv. 使用 numpy 矩阵运算简单实现了大致的 Bert 模型；

v. 学会在不同平台使用 GPU 和使用预训练模型进行迁移学习和参数微调。

b) 展望

Bert 和 Transformer 模型作为目前自然语言处理领域 SOTA 的模型，捕获序列能力强，虽然模型训练成本高，但是可以直接部署预训练模型进行迁移学习和参数微调，进行模型训练用于捕获序列特征，为下游任务进行训练。未来展望：多模态的自然语言处理的情感分析，是未来研究方向的蓝海之一（目前瓶颈：高质量的标注数据少）。

（以下 2022 年 5 月 24 日星期二 更新）

目前，GPT-2（General Pretrained Transformer 2.0）在自然语言处理领域使用也较多，GPT-1 推出早于 Bert，Google 推出的 Bert 模型是目的是为了对标 GPT-1 在 2019 年达到了 SOTA 的水平。GPT-2 是 GPT-1 的升级版，是生成式的语言预测模型，使用无监督的方式、40GB 的爬虫语料库、堆叠 48 层单向 Transformer 模块、扩大序列长度和网格参数级，因此，通过了 GLUE 标准的 7 个测试。此外，GPT-3 已经在 2020 年推出，GPT-2 和 GPT-3 都可以用于创造性写作（包括：新闻、小说、诗歌、歌曲等，甚至自动生成全栈代码），但是由于商业价值 OpenAI

和微软并没有开源。对于 Transformer 的未来展望：是在 GPT 系列基础上优化成通用 Transformer-based 预训练语言模型，并尝试解决语言重复和曝光问题。

六、 附录

a）结果截图

classification_report

	precision	recall	f1-score	support
0	0.90	0.91	0.90	2459
1	0.91	0.90	0.90	2541
accuracy			0.90	5000
macro avg	0.90	0.90	0.90	5000
weighted avg	0.90	0.90	0.90	5000

图表 2 TF-IDF + Logistic

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

```
======== 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 epcoh took: 0:04:30

Running Validation...
  Accuracy: 0.88
  Validation took: 0:00:20
```

```
======== 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 epcoh took: 0:04:28

Running Validation...
  Accuracy: 0.89
  Validation took: 0:00:20

Training complete!
```

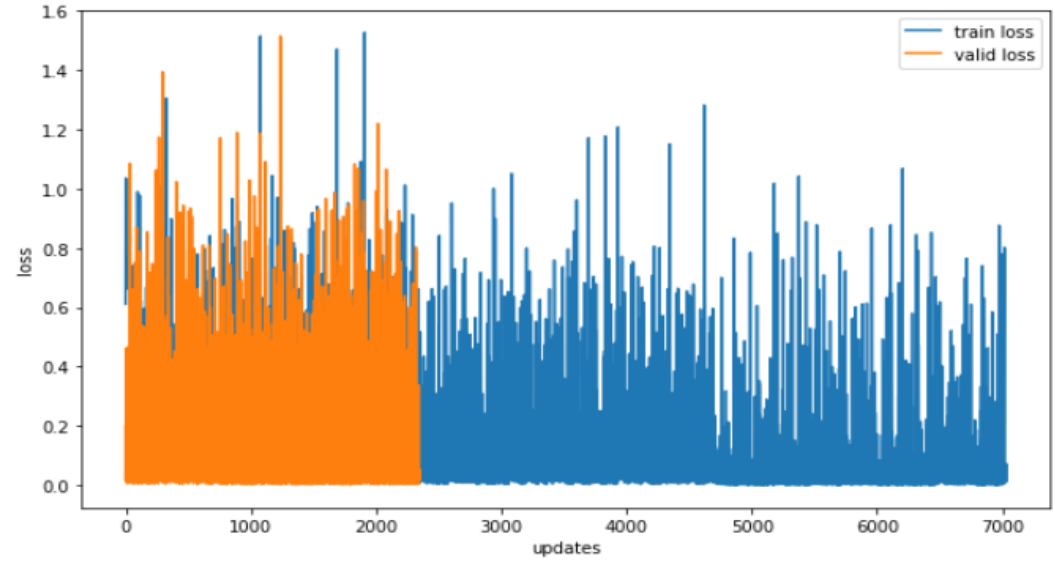
图表 11 Bert 训练

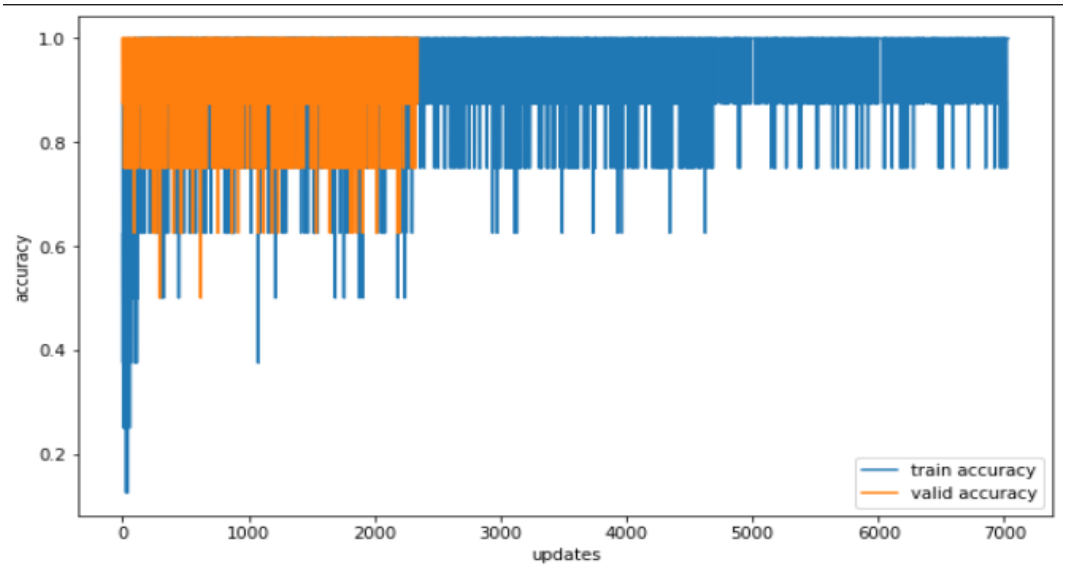
```
Training epcoh took: 0:04:47

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

Training complete!
```

图表 12 Bert 验证





图表 10 Transformer 损失图和准确率图（是收敛了，但是 figsize 太小了，所以很密集。）

b) 关键代码截图

```
word_vectorizer = TfidfVectorizer(sublinear_tf=True, # 将tf替换为1 + log(tf)
                                  strip_accents='unicode', # 在预处理步骤中删除重音符号并执行其他字符规范化
                                  analyzer='word', # 特征是由单词组成
                                  token_pattern=r'\w{1,}', # 等价于 '[^A-Za-z0-9_]'，至少匹配一次
                                  stop_words='english', # 直接过滤停用词
                                  ngram_range=(1, 1), # 仅使用双字符
                                  max_features=10000) # 只考虑根据语料库中的词汇频率排序的顶部
word_vectorizer.fit(all_text) # 从训练集学习词汇和idf
word_vectorizer
```

```python
def remove_emoji(text):
    '''使用正则表达式去除网页标记'''
    emoji_pattern = re.compile("["
                               u"\U0001F600-\U0001F64F"
                               u"\U0001F300-\U0001F5FF"
                               u"\U0001F680-\U0001F6FF"
                               u"\U0001F1E0-\U0001F1FF"
                               u"\U00002702-\U000027B0"
                               u"\U000024C2-\U0001F251"
                               "]+", flags=re.UNICODE)
    return emoji_pattern.sub(r'', text)
df['review']=df['review'].apply(lambda x: remove_emoji(x))
```

```python
%%time
from gensim.models import word2vec
np.set_printoptions(suppress=True)

feature_size = 256
context_size = 5
min_word = 1

word_vec= word2vec.Word2Vec(tokenized, vector_size=feature_size, \
                            window=context_size, min_count=min_word, \
                            epochs=50, seed=42)
```

Python

```
Wall time: 8min 23s
```

```python
transformer = transformers.AutoModel.from_pretrained(transformer_name)
```

Python

```python
transformer.config.hidden_size
```

Python

```python
class Transformer(nn.Module):
    def __init__(self, transformer, output_dim, freeze):
        super().__init__()
        self.transformer = transformer
        hidden_dim = transformer.config.hidden_size
        self.fc = nn.Linear(hidden_dim, output_dim)

        if freeze:
            for param in self.transformer.parameters():
                param.requires_grad = False

    def forward(self, ids):
        # ids = [batch size, seq len]
        output = self.transformer(ids, output_attentions=True)
        hidden = output.last_hidden_state
        # hidden = [batch size, seq len, hidden dim]
        attention = output.attentions[-1]
        # attention = [batch size, n heads, seq len, seq len]
        cls_hidden = hidden[:,0,:]
        prediction = self.fc(torch.tanh(cls_hidden))
        # prediction = [batch size, output dim]
        return prediction
```

Python

```
# For each epoch...
for epoch_i in range(0, epochs):

    # ========================================
    #               Training
    # ========================================

    # Perform one full pass over the training set.

    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')

    # Measure how long the training epoch takes.
    t0 = time.time()

    # Reset the total loss for this epoch.
    total_loss = 0
    model.train()
    for step, batch in enumerate(train_dataloader):

        if step % 40 == 0 and not step == 0:

            elapsed = format_time(time.time() - t0)

            print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {:}.'.format(step, len(train_dataloader), elapsed))

        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)

        model.zero_grad()
        outputs = model(b_input_ids,
                    token_type_ids=None,
                    attention_mask=b_input_mask,
                    labels=b_labels)

        # The call to `model` always returns a tuple, so we need to pull the
        # loss value out of the tuple.
        loss = outputs[0]
        total_loss += loss.item()
```

```
# ========================================
#               Validation
# ========================================
# After the completion of each training epoch, measure our performance on
# our validation set.

print("")
print("Running Validation...")

t0 = time.time()

# Put the model in evaluation mode--the dropout layers behave differently
# during evaluation.
model.eval()

# Tracking variables
eval_loss, eval_accuracy = 0, 0
nb_eval_steps, nb_eval_examples = 0, 0

# Evaluate data for one epoch
for batch in validation_dataloader:
    # Add batch to GPU
    batch = tuple(t.to(device) for t in batch)
    b_input_ids, b_input_mask, b_labels = batch
    with torch.no_grad():
        outputs = model(b_input_ids,
                        token_type_ids=None,
                        attention_mask=b_input_mask)

    # Get the "logits" output by the model. The "logits" are the output
    # values prior to applying an activation function like the softmax.
    logits = outputs[0]

    # Move logits and labels to CPU
    logits = logits.detach().cpu().numpy()
    label_ids = b_labels.to('cpu').numpy()

    # Calculate the accuracy for this batch of test sentences.
    tmp_eval_accuracy = flat_accuracy(logits, label_ids)

    # Accumulate the total accuracy.
    eval_accuracy += tmp_eval_accuracy

    # Track the number of batches
    nb_eval_steps += 1
```