

机器学习大作业设计报告

专 业 名 称: 计算机科学与技术学院

课 程 名 称: 机器学习

指 导 教 师 : 黄宏

学 生 学 号: U201911111

学 生 姓 名: 刘庆远

二〇二一年十一月

郑 重 声 明

本人呈交的设计报告，所涉及到的模型设计，代码工作，报告撰写，都是独立实验工作所取得的成果，所有数据、图片资料真实可靠。三个模型的实验代码已放置附件。

本人签名: 刘庆远

日期:2021.11.30

前言

首先，非常感谢老师与学长们也可为我们提供这样一个创意与趣味十足的大作业项目。从一开始的对于数据清洗的思考，对于停用词，标点是否保留的研究，再到文本翻译，文本词向量化，以及最后的模型的尝试与选择，在将近一周的探索中，经过不断的尝试与改进，我认为我学到的知识并不比理论课上学到的少，也掌握了一些真正可以应用与现实问题的知识。

因为之前有稍微了解过神经网络，以及图学习的一些模型，图学习模型的很多 idea 都是从 NLP 领域迁移过来的。所以看到大作业是 NLP 相关的任务，我脑海中第一个想出来的就是如何进行数据处理，如何得到一个好的 word embedding，因为神经网络训练并不会直接训练单词，而是先对于词语的进行 embedding，再将其输入神经网络。所以，在我看来，关键问题其一便是 embedding 的计算。然而在我查看原始数据集文本时，我发现其语言种类不同，对应的 unicode 编码肯定也不会一样，如果模型中同一个词因为语言问题而被赋予了不同的 embedding，我认为这个训练则毫无意义，所以我大作业的第一步便是文本翻译。

文本翻译完成之后，下一步便是清洗数据，使用停用词去掉对语义影响不大的单词与标点，然后将文本划分为词袋模型再去计算 word embedding。然而，仅仅 2200 个文本数据究竟能不能训练出来一个准确的 embedding 呢？我感到怀疑，虽说因为我们的的大作业是二分类，最终 embedding 稍微有点差错可能影响不大，但我还是希望模型能更加准确一些，通过网上查阅资料，我认为 Bert 架构为我提供了一个很好的参考，在 Bert 训练过程中，他采用了双向的 transformer 也就是一个 attention 机制来学习上下文，并为其他研究者提供预训练好的 pre-train 语言

模型。基于巨人的肩膀上，我再进行一个下游训练 Fine- train，就可以获得更好更准确的 embedding。

在获得了准确的 embedding 后，就是模型的选择了，在本次任务中我并没有选择单一一个模型，因为好奇心理，而且机器学习领域的算法实在是有点多，所以我希望探索更多的 NLP 处理方式。

第一个模型我选择的是朴素贝叶斯，在之前实验课实现 bayes 模型时，我发现当时所要求的 bayes 模型较为简单，有很多处考虑不全，比如在文本中对于出现次数为 0 的单词处理，以及拉普拉斯平滑等非常重要的机制都没有实现，所以上次实现的模型可能实际应用中并无价值。而且，在理论课上，我也了解到了关于 bayes 正确预测的机制，随着数据集的曾长，朴素 bayes 模型必然会趋于准确，我也很好奇在我们的任务中，仅仅 2200 个训练文本究竟可以达到怎样的效果。

第二个模型我选择的是随机森林模型，因为数据集的原因大多数模型可能无法较好的训练，而随机森林采用 Bagging 的集合策略使他的准确率得到保证，除此之外我对于他在数据集过小时避免有害数据的干扰能力也很好奇。

第三个模型我是基于 Bert 架构进行了一下改进实现做的，因为 Bert 论文所提到的模型并不适合长文本训练，所以我就想可不可以将我们的数据集进行一下调整，划分为小组字符（实验里我使用的长度为 200）分开使用 Bert 模型去计算，最终再采用一些手段进行 label 整合。在将模型进行 tokenize 化之后，就是对文本的学习了，因为我之前做过一些股票预测的模型，所以第一次时间就联想到了使用 LSTM 长段记忆神经网络，LSTM 是循环神经网络 RNN 的一个变种，这种神经网络在其神经元的阀门构造造就了其可以学习一个序列的数据。所以我就想使用 LSTM 来基于 embedding 进行文本分类是可行的。

我将在下文逐一介绍实验步骤与我的尝试。

目 录

1	数据预处理	1
1.1	数据结构	1
1.2	文本翻译	1
1.3	过滤数据	1
1.4	构建向量	2
1.4.1	CountVectorizer	2
1.4.2	TfidfVectorizer	3
2	Naive Bayes	4
2.1	模型机制简介	4
2.2	关于拉普拉斯平滑	5
2.3	模型实现	5
2.4	实验结果	5
3	随机森林模型	7
3.1	模型机制简介	7
3.1.1	bagging 算法	7
3.1.2	CART 决策回归数	8
3.1.3	RF 总流程	9
3.2	模型实现	10
3.3	实验结果	10
4	基于 Bert 与 LSTM 模型	12
4.1	实验平台	12
4.2	模型介绍	12

4.2.1	项目简介	12
4.2.2	模型框架	12
4.3	模型搭建	13
4.3.1	pre-train 模型获取	13
4.3.2	数据处理	13
4.3.3	利用 transformer 进行 fine-train 训练	14
4.3.4	WE 精准度	15
4.3.5	搭建 LSTM 神经网络	15
4.4	实验结果	16
4.5	对本模型思考	16
5	评估标准	17
5.1	f1-score	17
6	思考与 future work	18
	参考文献	19
	附录 A 实验代码	20

1 数据预处理

1.1 数据结构

在本次试验中，我使用的读取文件方法就是关卡所给的 json 读取方法，具体实现在 file 模块里，本次试验所有操作均是在一个列表中进行，列表每一个元素为一个字典，字典内含有 content 与 label 两个键值，实验中所有的数据结构均来自于初始列表。

1.2 文本翻译

一开始我是使用爬虫，也就是 Requests 包来尝试向翻译网站抓包，希望可以复习一下爬虫技能。目标网站就是百度翻译，但一旦数据变长，比如一次几千个字符，或者访问频率过快，就出错，可能是网站的反扒机制吧。所以我就改用了百度翻译 API，通过 GET 请求就可以调用此接口。

再次翻译过程中有两点时需要注意的：

1. 百度翻译 API 普通版有访问频率限制，所需需要每请求一次暂停一下
2. 百度翻译单次翻译数据最长为 500 字符，所以还需要将文本切块翻译，接收数据后整合。
3. 在翻译时需要制定原语言语种，其实百度翻译请求报文这一栏是可以填 auto 的，这也节约了许多时间。

因为翻译速度关系，我向其他人又借了 3 个账号，这样翻译速度又提升了 3 倍，最终，训练集与测试集数据在 2h 内被全部翻译。

1.3 过滤数据

在刚被爬取的文本数据中，有着种种杂乱的标签，符号，乱码等等，虽说数据量不大，但这些数据不仅会影响翻译文本，而且在训练数据集相对较少的情况下对模型的训练有着毒害的作用。所以数据处理的第一步便是数据过滤，在我看来，

需要过滤的数据有标点符号,html 标签,数字,停用词以及转换大小写

在我所设计的模型内,逻辑回归,Naive Bayes,Bert 模型共用的一套数据集过滤系统。在 Naive Bayes 项目里 utils 模块便是负责数据过滤的。

Step 1 调用 utils 模块下的 gettext 函数,取得去除文本中 html 标签的文本。

Step 2 调用 utils 模块下的 text_process 函数,去除文本中标点符号和数字内容。

Step 3 调用 utils 模块下的 getword 函数,将文本进行分词并进行大小写转换。

Step 4 使用 nltk 库里的停用词列表,在分词时逐一判断,筛选出在 stop words 里面的词。

1.4 构建向量

在 Naive bayes 与逻辑回归项目中,我使用的是 sklearn 包进行词向量训练,就是基于词频与词重要性的,与在 Bert 模型里进行 fine-train 的并不一样。当然 Bert 改进模型的具体训练过程我会在下文说明。

基于 sklearn 词向量学习主要基于以下两个模块:

CountVectorizer	统计词频
-----------------	------

TfidfVectorizer	重要性评估
-----------------	-------

1.4.1 CountVectorizer

CountVectorizer 会将文本中的词语转换为词频矩阵,它通过 fit 函数计算各个词语出现的次数。CountVectorizer 通过计数来将一个文档转换为向量。当不存在先验字典时,CountVectorizer 作为 Estimator 提取词汇进行训练,并生成一个 CountVectorizerModel 用于存储相应的词汇向量空间。该模型产生文档关于词语的稀疏表示,其表示可以传递给其他算法。

假设 list 里面有 a 篇文章,文章里面预计有 b 种类的单词,那么 fit 函数会把原先的 list 元素进行处理。将这些单词进行处理。处理出来的是一个稀疏矩阵,代表每个单词在每篇文章中出现的次数。举个例子: $A[i, j] = P$ 代表第 i 篇文章的第 j 个词语出现了 P 遍。

使用词频向量化, 就可以很快地提取中隐含在文章中的词语特征, 当然这个特征仅仅是基于词频。我们将此词频矩阵成为 TF

1.4.2 TfidfVectorizer

然而, 仅仅通过词频向量化仍会讲一些任务无关性的词语带入进来, 并赋予他们和关键词一样的权重, 比如在本次试验中, 新闻情感分析中 bad, good 等词的比重理应要比一些名词比如 banana 要大, 故 TfidfVectorizer 引入了一个类似于 attention 的机制, 来为每个词富裕任务重要性的权重。其计算公式为:

$$\log(\frac{n}{1 + df})$$

其中 n 为文档总数, df 为含有所计算单词的文档数量, df 越小, 就是说出现频率越小的单词意义越大

最终将两者训练出来的词频矩阵 TF 与逆反文档频率 IDF 相乘, 得出真正可用于训练的词向量表示。

2 Naive Bayes

2.1 模型机制简介

贝叶斯方法是一个历史悠久，有着坚实的理论基础的方法，同时处理很多问题直接而又高效，很多高级自然语言处理模型也可以从它演化而来。因此，在本次试验中，我认为朴素贝叶斯会有着不俗的效果。

但朴素 bayes 模型仍有着一些缺点，就是朴素 bayes 的训练和语序无关，在一些比较复杂的分类任务中，可能会因为这种语序无关性而很低效。

原始 bayes 公式

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

朴素 bayes 的理论部分并不难理解，再此公式中，我们只需要将 X 视为特征，Y 视为标签，则就会很好理解， $P(Y|X)$ 就代表着拥有者 X 特征的文本是 Y 标签的概率。再次公式中， $P(X) P(Y)$ 代表着某种标签或者某种特征出现的概率，如果在 $P(X)$ 中 X 为多维，在其特征之间无关的情况下，则可以认为

$$P(\mathbf{X}) = \prod_{i=1}^n P(x_i)$$

对于 $P(X|Y)$ 也是同理，此式含义为在标签为 Y 的条件下，X 特征出现的概率，如果 X 为一个多为特征，与上式同理，我们可以得到：

$$P(\mathbf{X}|Y) = \prod_{i=1}^n P(x_i|Y)$$

对于每一个 $P(x_i|Y)$ 其代表着在标签 Y 的条件下， x_i 特征的出现概率。设置集合 $n_i^Y = \{1, 0, 3, 5, 2, \dots\}$ ，其中 $\sum_{a=0}^n n_i^Y = m_i^Y$ ，其含义为 x_i 特征在每个标签为 Y 的文本中出现的频率，故，关于 $P(x_i|Y)$ 具体计算公式为：

$$P(x_i|Y) = \frac{m_i^Y}{\sum_{i=0}^n m_i^Y}$$

在先验概率和后验概率都计算完成之后，理应就可以带入朴素 bayes 公式进行文本分类任务了，这也是上一次课上实验我们所完成的内容。但是仍有需要考

虑的问题，其一就是拉普拉斯平滑，具体说就是，如果某一维特征在标签 Y 中并没有出现过，这就回导致对应的概率值为 0，在模型预测的过程中，将会仅仅因为这一个特征导致整体预测概率变为 0，这会是一个非常愚蠢的错误。

2.2 关于拉普拉斯平滑

所以，为了避免这种情况，在计算标签 Y 的条件下， x_i 特征的出现概率 $P(x_i|Y)$ 时，我们引入拉普拉斯平滑：

$$P(x_i|Y) = \frac{m_i^Y + \alpha}{\sum_{i=0}^n m_i^Y}$$

公式中 α 便是引入的常数，这样即便是某一位特征出现概率为 0，在实际计算时也会将其转变为一个很小的值，避免了个别特征对模型干扰的问题。

2.3 模型实现

在 bayes 模型试验中，我选择的是 sklearn 库里面的 MultinomialNB 先验多项式分布的朴素 bayes 模型。具体实现方式如下：

1. 数据预处理，如第一章所述
2. 使用 TfidfVectorizer 进行特征提取，将文本特征量化
3. 调用 MultinomialNB 传入文本特征与 label 进行 fit
4. 使用 MultinomialNB 模型对测试集进行预测

具体实现如下图：

```
tfidf_vector.fit(train_X)
train_tfidf = tfidf_vector.transform(train_X)
test_tfidf = tfidf_vector.transform(test_X)
clf_nb = MultinomialNB(alpha=j) # 模型参数可以根据分类结果进行调优
clf_nb.fit(train_tfidf, train_y) # 模型训练
y_pred = clf_nb.predict(test_tfidf) # 模型预测
```

图 2.1 模型实现

2.4 实验结果

为了能够更加准确的展现出拉普拉斯平滑对于预测结果的影响，也为了调试出一个更好的模型，我将训练机分为了两部分，一部分用于训练，另一部分用于

验证 α 值对于 accuracy 的影响，基于控制变量原则，训练集与验证集部分保持不变。实验结果如下图所示。从公式计算来看，二分类问题 accuracy 与 microf1 值相等

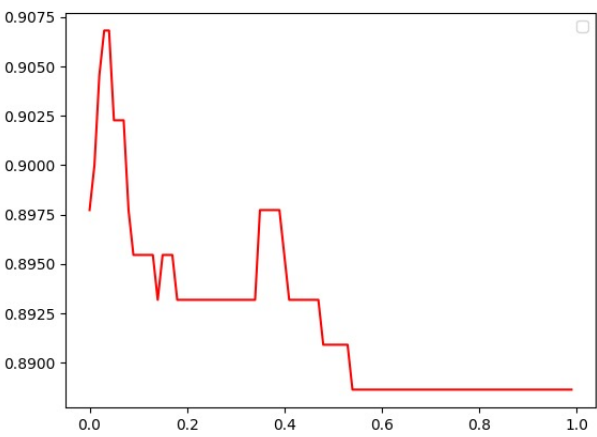


图 2.2 α 值与 microf1 与 accuracy 关系

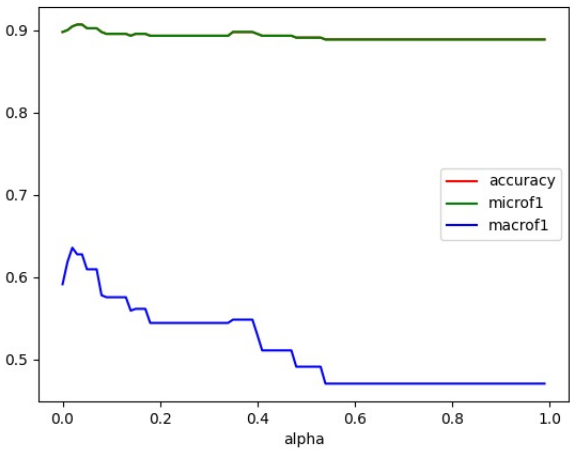


图 2.3 α 值与 macrof1 与 accuracy 关系

综上，将 α 值设置为 0.04 左右效果最佳。

3 随机森林模型

3.1 模型机制简介

随机森林是一种集成算法 (Ensemble Learning)，它属于 Bagging 类型，通过组合多个弱分类器，最终结果通过投票或取均值，使得整体模型的结果具有较高的精确度和泛化性能。其可以取得不错成绩，主要归功于“随机”和“森林”，一个使它具有抗过拟合能力，一个使它更加精准。

3.1.1 bagging 算法

Bagging 也叫自举汇聚法 (bootstrap aggregating)，是一种在原始数据集上通过有放回抽样重新选出 k 个新数据集来训练分类器的集成技术。它使用训练出来的分类器的集合来对新样本进行分类，然后用多数投票或者对输出求均值的方法统计所有分类器的分类结果，结果最高的类别即为最终标签。此类算法可以有效降低 bias，并能够降低 variance。

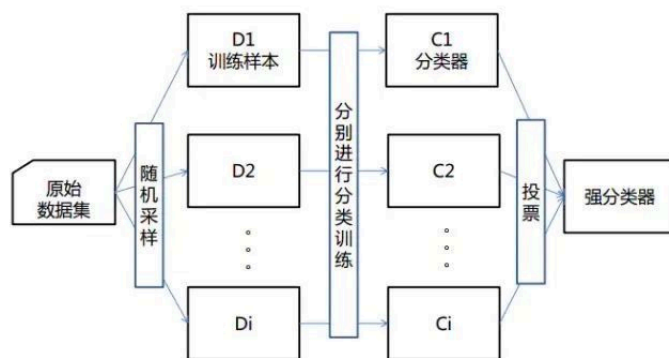


图 3.1 bagging 算法示意图

随机森林 (Random Forest, RF) 是 Bagging 算法的一种，RF 相对于 Bagging 只是对其中一些细节做了自己的规定和设计。

- **弱分类器**: 首先，RF 使用了 CART 决策树作为弱学习器。换句话说，其实我们只是将使用 CART 决策树作为弱学习器的 Bagging 方法称为随机森林。

- **随机性:** 同时，在生成每棵树的时候，每个树选取的特征都仅仅是随机选出的少数特征，一般默认取特征总数 m 的开方。而一般的 CART 树则是会选取全部的特征进行建模。因此，不但特征是随机的，也保证了特征随机性。
- **样本量:** 相对于一般的 Bagging 算法，RF 会选择采集和训练集样本数 N 一样个数的样本。
- **特点:** 由于随机性，对于降低模型的方差很有作用，故随机森林一般不需要额外做剪枝，即可以取得较好的泛化能力和抗过拟合能力 (Low Variance)。当然对于训练集的拟合程度就会差一些，也就是模型的偏倚会大一些 (High Bias)，仅仅是相对的。

3.1.2 CART 决策回归数

在随机森林中，为了实现 bagging 算法，首先需要定义一个分类模型，目前普遍采用的便是 CART 决策回归树作为其内部分类模型

首先，特征选择，CART 分类树算法使用基尼系数选择特征，基尼系数代表了模型的不纯度，基尼系数越小，不纯度越低，特征越好。这和信息增益（率）相反。

基尼系数

$$GINI(f) = \sum_{i=1}^m f_i(1 - f_i)$$

其中 f_i 代表在相同的特征下不同 $label_i$ 出现的频率， m 代表 label 总数。

一个数据集中，某一维包含 n 个特征值的特征 f 的纯度可用基尼值来度量，其中每个特征值占比设为 $P(f^k)$:

$$GINI_{Gain}(f) = \sum_{k=1}^k P(f^k) \sum_{i=1}^m f_i^k(1 - f_i^k)$$

基于 $GINI_{Gain}(f)$ 的定义，最好的划分为使得 $GINI_{Gain}(f)$ 最小的划分。

CART 就是通过这种方式，不断的递归寻找这棵树的特征子集的所有可能的分割点，寻找 Gini 系数最小的特征的分割点，将数据集分成两个子集，直至满足停止条件为止。**剪枝**避免决策树过拟合样本。前面的算法生成的决策树非常详细并且庞大，每个属性都被详细地加以考虑，决策树的树叶节点所覆盖的训练样本

都是“纯”的。因此用这个决策树来对训练样本进行分类的话，你会发现对于训练样本而言，这个树表现完好，误差率极低且能够正确得对训练样本集中的样本进行分类。训练样本中的错误数据也会被决策树学习，成为决策树的部分，但是对于测试数据的表现就没有想象的那么好，或者极差。

3.1.3 RF 总流程

首先，RF 使用了 CART 决策树作为弱学习器。第二，在使用决策树的基础上，RF 对决策树的建立做了改进，对于普通的决策树，我们会在节点上所有的 n 个样本特征中选择一个最优的特征来做决策树的左右子树划分，但是 RF 通过随机选择节点上的一部分样本特征，这个数字小于 n ，假设为 n_{sub} ，然后在这些随机选择的 n_{sub} 个样本特征中，选择一个最优的特征来做决策树的左右子树划分。这样进一步增强了模型的泛化能力。

如果 $n_{sub} = n$ ，则此时 RF 的 CART 决策树和普通的 CART 决策树没有区别。 n_{sub} 越小，则模型约健壮，当然此时对于训练集的拟合程度会变差。也就是说 n_{sub} 越小，模型的方差会减小，但是偏倚会增大。在实际案例中，一般会通过交叉验证调参获取一个合适的 n_{sub} 的值。

1. 输入为样本集 $D = (x_1, y_1), (x_2, y_2), \dots, (x_i, y_i)$ ，弱分类器迭代次数 T 。
2. 输出为最终的强分类器 $f(x)$
3. 对于 $t=1, 2, \dots, T$: a) 对训练集进行第 t 次随机采样，共采集 m 次，得到包含 m 个样本的采样集 D_t b) 用采样集 D_t 训练第 t 个决策树模型 $G_t(x)$ ，在训练决策树模型的节点的时候，在节点上所有的样本特征中选择一部分样本特征，在这些随机选择的部分样本特征中选择一个最优的特征来做决策树的左右子树划分
4. 如果是分类算法预测，则 T 个弱学习器投出最多票数的类别或者类别之一为最终类别。如果是回归算法， T 个弱学习器得到的回归结果进行算术平均得到的值为最终的模型输出。

3.2 模型实现

本次试验所使用的随机森林模型，我采用的是 sklearn 库里的 RandomForestClassifier 将预处理好的模型首先送入 CountVectorizer 进行词向量化，

```
# Initialize a Random Forest classifier with 100 trees
forest = RandomForestClassifier(n_estimators=100)
forest = forest.fit(train_data_features, clean_train_label)
```

图 3.2 RF 模型代码实现

3.3 实验结果

我首先观察了随着 CART 决策树的增加，随机森林 accuracy 与 macrof1 值的变化，发现在 tree_num 较小的时候，准确率和 num 数量成正比，但在大于 100 时，正确率将不再变化趋于平稳。

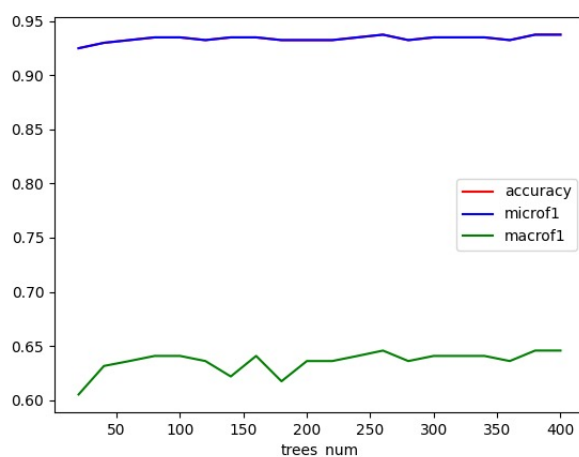


图 3.3 Rtree_num 与 accuracy, macrof1 关系

从上述模型效果来看在 tree_num 较小的时候，准确率和 num 数量成正比，但在大于 100 时，正确率将不再变化趋于平稳。

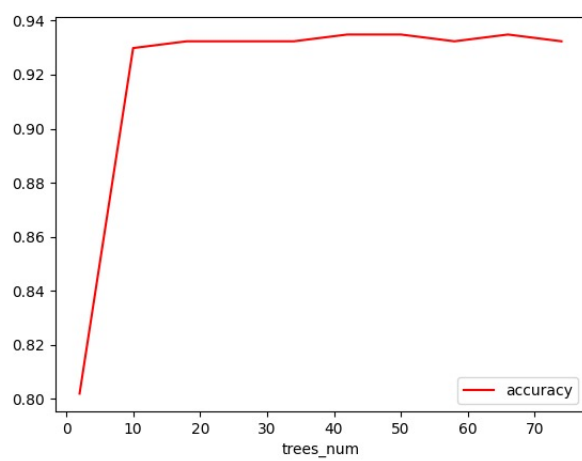


图 3.4 Rtree_num 小于 100 时与 accuracy 关系

4 基于 Bert 与 LSTM 模型

4.1 实验平台

本项目首先使用 Bert 模型计算 word embedding，之后使用 LSTM 神经网络学习词向量化的文本内容。因为计算量较大，涉及的库较多，我使用了 Google colab 作为实验平台。

4.2 模型介绍

4.2.1 项目简介

- **1.pre-train 模型获取:** 因为原始 Bert 模型并不支持长文本数据训练，所以为了更好的适应本次作业，我采用的 Hugging Face 组织下的 transformer 作为轮子构造 Bert 模型。当然 Hugging Face 也提供了预训练好的 Bert 模型，我使用的是 bert-base-uncased 模型
- **2. 项目难点:** 本次难点有二：第一，Bert 原始模型并不支持长文本数据输入，那我们应该如何才能使用 Bert 呢？第二，Bert 仅是一个文本词语特征提取的一个模型，在提取完特征之后我们该是用什么分类模型达到较好的效果呢？

4.2.2 模型框架

- 1. 数据预处理
- 2. 文本分割适应 Bert 接口
- 3. 基于 pre-train model 使用 Bert 进行 fine- train
- 4. 评估 fine-train model
- 5. 搭建 LSTM 神经网络
- 6. 使用 fine-train model 进行词向量化
- 7. 使用 LSTM 进行测试及数据预测

4.3 模型搭建

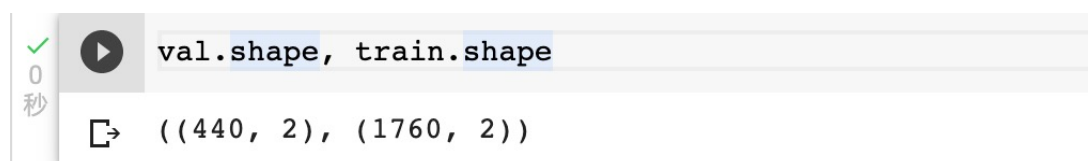
4.3.1 pre-train 模型获取

本模型所采用的 pre-train model 来自于 Hugging Face 的 bert-base-uncased 模型，hugging face 开源的 transformer 库为我的模型构建提供了许多便利，所以我使用的也是他们的预训练模型。

4.3.2 数据处理

数据处理方面，首先是使用翻译好的文本作为原始数据。将翻译后的数据读入之后，利用正则表达式将标点符号里面的逗号去字母和数字与标点无关符号全部去除。这样文本既保留了一些重要的符号信息，也有正常的文本数据。

在对文本符号进行删除之后，就对训练集 = 集与验证集进行划分，我选择的是验证集占比为 0.2。此式文本格式为：



```
val.shape, train.shape
```

```
((440, 2), (1760, 2))
```

图 4.1 数据集与验证集

在对训练集与测试集划分之后，就要对文本进行分割了。在 Bert 模型中，因为其算法的特殊性，随着输入文本长度增长，其训练时间会随之指数级增长，所以 Bert 模型限制最大输入长度为 512 字符，但本次任务中，基本都是长文本数据集，所以需要划分文本，将每个文本划分为一小块一小块输入 Bert 模型进行 fine-train，在训练完成后重新进行整合。因为在输入 Bert 模型后，每个单词 tokenizer 之后也有可能被分成好几部分。所以实际可输入的句子长度可能会增常，保险起见我将文本一次分割为最大长度为 200 字符的小块，再为每个小块配置标签。

具体转换如下：在训练好每个句子的 embedding 之后，将其句子组合起来，label 放置最后用于接下来 LSTM 模型的分类任务

句子1_a的embedding	label1
句子1_a的embedding	label1
句子1_a的embedding	label1
句子2_a的embedding	label2
句子3_a的embedding	label3
句子3_a的embedding	label3

句子1_a的embedding, 句子1_a的embedding, 句子1_a的embedding	label1
句子2_a的embedding	label2
句子3_a的embedding, 句子3_a的embedding	label3

4.3.3 利用 transformer 进行 fine-train 训练

```
[31] # 1.载入预训练模型
args={"model_name_or_path": "/content/drive/MyDrive/model/",
      "config_name": "/content/drive/MyDrive/model/",
      "tokenizer_name": "/content/drive/MyDrive/model/",
      }

config_class, tokenizer_class = MODEL_CLASSES["bert"]
model_class=BertForClassification

config = config_class.from_pretrained(
    args["config_name"],
    finetuning_task="",
    cache_dir=None,
)
tokenizer = tokenizer_class.from_pretrained(
    args["tokenizer_name"],
    do_lower_case=True,
    cache_dir=None,
)
model = model_class.from_pretrained(
    args["model_name_or_path"],
    from_tf=bool(".ckpt" in args["model_name_or_path"]),
    config=config,
    cache_dir=None,
)

model.to("cuda")
```

图 4.2 载入预训练模型

再将本地的 train_InputExamples 训练数据准备好之后，就开始用 Bert 模型训练本次文本的 embedding。

4.3.4 WE 精准度

在经过 Bert 计算出 word embedding 之后，我们需要验证一下我们所做的 fine- train model 正确性，这里我就使用了 Bert 模型做了一个简单的 accuracy 测量，同时，使用 Bert 做预测则代表着我的文本数据此时并没有合并，所以这不能代表模型最终效果，只能说 word embedding 正确性得到了保证。

在我随机划分的占比 0.2 验证集中预测结果如下：

```
# 4. 评估结果
results,eval_loss = evaluate(model, tokenizer, eval_dataset)
print()
print("Accuracy: ",results, "Loss: ",eval_loss)

Evaluating: 100%|██████████| 105/105 [00:41<00:00, 2.50it/s]
Accuracy: 0.8880239520958084 Loss: 0.3891933720248441
```

图 4.3 基于训练后的 WE 对分割后的文本进行分类结果

4.3.5 搭建 LSTM 神经网络

因为 LSTM 长段记忆神经网络是 RNN 循环神经网络的一个变种，其特性之一就是可以学习一个序列的数据，而 LSTM 优秀之处就在一它的阀门机制，他可以控制阀门来控制序列之前的数据在训练中的权重。所以我认为 LSTM 理应适应我们的文本分类作业，在本次试验中，我使用 tensorflow 的 keras 来进行 LSTM 网络的搭建。实现如下图：

Layer (type)	Output Shape	Param #
text (InputLayer)	[(None, None, 768)]	0
masking_2 (Masking)	(None, None, 768)	0
lstm_2 (LSTM)	(None, 100)	347600
dense_4 (Dense)	(None, 30)	3030
dense_5 (Dense)	(None, 10)	310
Total params: 350,940		
Trainable params: 350,940		
Non-trainable params: 0		

图 4.4 LSTM 模型结构图

在搭建好 LSTM 模型之后，将处理好的文本 embedding 送入 LSTM 模型进行训练，得到一个基于 Bert 词向量化之后量身定制的一个文本分类的模型。

4.4 实验结果

然而在我搭建好 LSTM 模型之后，使用组合后的数据训练完成，再去预测验证集与测试集时，发现模型能力明显不足，其 accuracy 仅有 10% 左右，我认为是我模型配置在哪出了问题或者是数据集过小的缘故？按理来说在测试 WE 精准度的时候就可以证明 WE 学习的还算不错，我目前还不是很明白，模型代码我放到了 GitHub 仓库里，如果学长或老师感兴趣可以帮我分析分析。

4.5 对本模型的思考

首先，我认为我对将文本划分之后将处理好的数据送入 Bert 训练是较为成功的，取得了相对不错的效果，划分后的文本数据使用 Bert 内置的分类器有将近 90% 的准确率。

LSTM 模型的搭建预测结果虽说出了一点问题，具体分析：

- (1) 不过我认为因该是我对于数据特征的，神经网络层数与每层神经元的研究不够透彻，也许在提取特征时不够全面。也就是说 LSTM 模型构造有一些问题。
- (2) 或许是将文本分类再送入 Bert 模型训练虽说可以提取到句子文本的信息，但是组合起来之后 LSTM 模型无法正确识别整个文本的信息，也就是说明我的这种将长文本划分分别塞入 bert 训练的方式并不可行，可能需要换一种方式计算 WE
- (3) 还有一种可能就是数据预处理有些许问题，但我目前没想好

但不管如何，毕竟最重要的是求解文本的 embedding，将 embedding 求出来再结合成为总的文本 embedding 就已经达成我的目标了。再往上层其实用哪个模型都可以进行分类。只不过考虑到更多，更复杂的数据我认为还是神经网络具有更好的效果，所以我才选择了 LSTM 模型进行分类预测。总的来说，我认为本次试验还算是成功，但未来仍需要继续研究 LSTM 问题出在哪里。

5 评估标准

5.1 f1-score

定义:

TP: True Positive 真阳性: 预测为正, 实际也为正 FP: False Positive 假阳性: 预测为正, 实际为负 FN: False Negative 假阴性: 预测为负, 实际为正 TN: True Negative 真阴性: 预测为负, 实际也为负. 接着我们可以根据这些概率统计的术语进行推导:

精准度 (precision): 指被分类器判定正例中的正样本的比重

$$pre = \frac{TP}{TP + FP}$$

召回率 (recall): 指的是被预测为正例的占总的正例的比重

$$rec = \frac{TP}{TP + FN}$$

另外, 介绍一下常用的准确率 (accuracy) 的概念, 代表分类器对整个样本判断正确的比重.

$$acc = \frac{TP + TN}{TP + TN + FP + FN}$$

根据精准度和召回率我们可以给出 micro-f1 分数:

$$micro - f1 - score = 2 * \frac{pre * rec}{pre + rec}$$

至于 macro-f1 分数:

$$macro - f1 = \frac{\sum_{i=1}^n micro - f1_i}{n}$$

此式中 n 为标签数量

6 思考与 future work

首先，我的观点是理论上来说在这次大作业里，理论上三个模型的分类效果应该差不多。虽说实际中朴素贝叶斯模型在处理小文本上可能会更具有优势，但我认为只要在保证 WE 一样的前提下神经网络也可以取到不错的效果。

随机森林算法在处理小数据集上的性能与朴素贝叶斯模型类似，这两个模型适合处理小组数据的原因我认为是速度更快，更轻量化，其效果较好的原因也是能较为合适的被本次任务数据所训练。

对于使用 Bert 模型也是建立在我对 Bert 训练有了一个初步的了解之后，我才开始计划因一种非常简单的长文本截断方式，分批进行 WE (word embedding 训练) 但因为我构建模型时自身的种种问题，导致正确率并不理想，我打算在接下来的时间里继续研究究竟是哪一步环节出现了，也会继续思考问题 Bert 与 LSTM 结合的模式。

对于使用 Bert 进行长文本分类，我在做完实验后才发现目前学术界与工业界也有了相关的研究，比如清华和阿里 20 年 paper: COGLTX 模型，并且目前也有很多关于长文本截断的手段，比如说池化法，压缩法，pooling 法。不过因为精力有限，我在进行实验之前并没对此进行研究，也算是本次试验的一个遗憾吧。

在本次试验中，因为时间原因，我仍有许多 idea 与计划没有实现，比如使用 Bert 训练后的 embedding 上在进行随机森林与朴素贝叶斯模型的训练，而非是仅仅使用了词频逆反文档率矩阵作为 word embedding，效果会如何。再比如如果使用目前的更为成熟的清华的 COGLTX 框架会怎样。以及对于文本截断方式的更多的探索，也许需要未来有缘再去研究了。

参考文献

- [1] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
- [2] Hierarchical Transformers for Long Document Classification
- [3] CogLTX: Applying BERT to Long Texts

附录 A 实验代码

实验代码：(1) 三个实验的模型代码均在压缩包内，每隔模型均有 README，特别是第三个，请按 README 操作不然可能会覆盖 fine- train 的模型。(2) 实验中文本路径可能会有些不协调，因为模型三我是在 Google colab 上做的，稍加改动就可以了。(3) 并没有每个模型里都包翻译程序，只有在随机森林林里有，因为翻译好的文本我就直接 copy 到下一个模型了