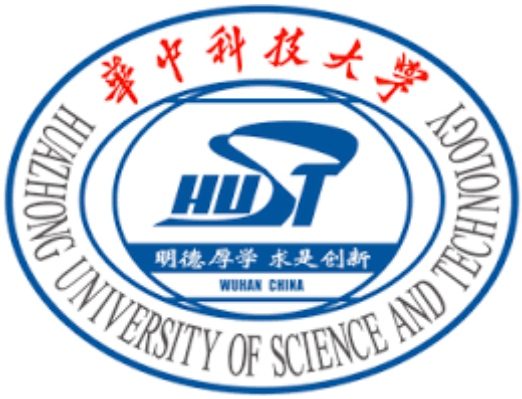
华中科技大学计算机科学与技术学院

机器学习：基于机器学习的新闻数据情感分析



专 业： 计算机科学

班 级： CS1906

学 号： U201915033

姓 名： 谭 璇

成 绩：

指导老师： 黄 宏

完成日期： 2020年11月29日

一、引言与实验题目

在本次课题中，我们要根据新闻的情感对新闻进行分类，新闻数据分为主观和客观两种情感，分别用1和0表示，即一个二分类问题。实验中通过训练集数据对模型进行训练，再通过模型预测测试集的label。将预测的label放入educoder平台进行计算，返回micro-f1值和macro-f1。本次实验的主要工作如下：

- **清洗数据**
- **将文本转化为词向量**
- **预测文本的情感**
- **对预测模型进行评价**

二、模型与算法设计

2.1 数据清洗

数据清洗是整个数据分析过程的第一步，就像做一道菜之前需要先择菜洗菜一样。 数据分析师经常需要花费大量的时间来清洗数据或者转换格式，这个工作甚至会占整个数据分析流程的80%左右的时间。

文本翻译

在本次课题中，数据集分为train.json和test.json两个不同的json文件。train.json为训练集，test.json为测试集。由于train.json和test.json中的数据是多种不同的语言组成的，我们先要将其统一翻译为英文。代码中调用google_trans_new包。代码片段如下所示：

```
res_content[index] = t.translate(text=content[index], lang_tgt='en')
```

在调用API进行翻译的时候要注意两个问题：

- google_train_new API一次只能翻译少于5000字节。因此，当遇到大于5000词的文本时，我们要先将其切割再调用API翻译再组装。
- 一次性调用多次API也就是访问多次translate.google.cn会导致判定流量异常，因此，我们需要访问一定次数之后切换IP再继续访问，具体可以使用VPN实现。

  具体代码如下所示：

```
for index in range(0, 300):
    print(index)
    if len(content[index]) >= 5000:
        str = textwrap.wrap(content[index], 2000)
        for i in range(len(str)):
            t = google_translator(timeout=10)
            str[i] = t.translate(text=str[i], lang_tgt='en')
        temp = '\n'
        res_content[index] = temp.join(str)
    else:
        t = google_translator(timeout=10)
        res_content[index] = t.translate(text=content[index], lang_tgt='en')
    print(res_content[index])
```

文本清洗

文本数据中有一些对新闻情感没有影响的字符，如果让大量这样的字符存在于数据中，会对训练和测试结果造成影响，因此我们需要删除这些字符，主要有两种字符需要删除,一种是标点符号，另外一种是停用词。

- **删除标点符号**

  我们直接使用正则表达式删除标点符号，代码如下图所示：

```
dic['content'] = dic['content'].replace('[,\.!?:()"]', '')
```

- **删除停用词**

  我们通过直接使用nltk包中的stopwords，核心代码如下所示：

```
nltk.download("stopwords")
WPT = nltk.WordPunctTokenizer()
stop_word_list = nltk.corpus.stopwords.words('english')
dic['content'] = word_tokenize(dic['content'])
temp = [word for word in dic['content'] if word not in list(stop_word_list)]
```

通过翻译文本和删除无关字符，我们最终得到了完备的、干净的数据。

2.2 基于Keras的模型与算法设计

构建模型与预测

- **拆分训练集**

  首先，我们将原始数据按照比例分割为“测试集”和“训练集，运用train_test_split()函数，具体代码如下：

```
X_train,X_test, y_train, y_test
=sklearn.model_selection.train_test_split(train_data,train_target,test_size=0.4,
random_state=0,stratify=y_train)
```

  我们将test_size设置为0.4，即训练集中的测试集为占比0.4。

  接着，我们将高频词设置为15000，根据文本列表更新数据，并将文本中的每个文本都转换为整数序列。

```
tokenizer = Tokenizer(num_words=15000)  # indicate the quantity of vocabulary
tokenizer.fit_on_texts(data)  # convert data from natural language to list of index of
word's position in the vocabulary
x_train_tokens = tokenizer.texts_to_sequences(x_train)
x_test_tokens = tokenizer.texts_to_sequences(x_test)
```

- **初始化模型**

  我们运用Sequential函数初始化模型，并将embedding_size值设置为50，表示词向量大小为50维。

  第一层为embedding层，每个句子是大小为15000的数组。代码如下：

```
model.add(Embedding(input_dim=15000, output_dim=embedding_size, input_length=max_tokens,
name='embedding_layer'))
```

  第二层包含8个LSTM 单元，代码如下：

```
model.add(LSTM(units=8, return_sequences=True))
```

  第三层包含4个LSTM单元，代码如下：

```
model.add(LSTM(units=4))
```

- **训练模型**

  设置validation_split=0.2，epochs=5，batch_size=4000，核心代码如下：

```
history = model.fit(x_train_pad, y_train, validation_split=0.2, epochs=5,
batch_size=4000, shuffle=True, verbose=1)
```

模型优化

在实际测试中我们发现测试的结果正确率不高，因此我们要对参数进行调节，优化准确率。我们主要对以下参数进行调节：

- fit函数中的epochs参数，epoch数是一个超参数，它定义了学习算法在整个训练数据集中的工作次数。一个Epoch意味着训练数据集中的每个样本都有机会更新内部模型参数。通过调节这个参数，从而改变模型过拟合或者欠拟合的状态。我们一开始将epochs参数设置为5,这就会造成过拟合的状态，因此我们不断调节epochs,达到最佳结果。

2.3 基于Naïve Bayes的模型与算法设计

朴素贝叶斯法（Naïve Bayes）是基于贝叶斯定理与特征条件独立假设的分类方法，属于统计学分类方法。简单来说，朴素贝叶斯分类器假设在给定样本类别的条件下，样本的每个特征与其他特征均不相关，对于给定的输入，利用贝叶斯定理，求出后验概率最大的输出。朴素贝叶斯法实现简单，学习与预测的效率均较高，在文本分类领域有广泛的应用。

文本数据的一个典型特征就是其维度较大，比如一篇文档，会有几千甚至上万个词，但是不同类型或主题的文档所用词汇差距较大，可以不考虑词汇出现的顺序，即采用bag of words模型，假设文本中每个词的出现都是独立的。基于此类假设的文本分类问题，可以采用朴素贝叶斯方法进行求解。

构建模型与预测

- **CountVectorize使用总结**

CountVectorizer是属于常见的特征数值计算类，是一个文本特征提取方法。对于每一个训练文本，它只考虑每种词汇在该训练文本中出现的频率。

CountVectorizer参数如下代码所示：

```
CountVectorizer(input='content', encoding='utf-8',  decode_error='strict',
strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, stop_words=None,
token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), analyzer='word', max_df=1.0,
min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<class 'numpy.int64'>)
```

我们将除max_features之外的所有代码设置为默认，调节参数max_features。代码如下所示：

```
vector = CountVectorizer(max_features=4000)
```

在课题的代码中，CountVectorizer会将文本中的词语转换为词频矩阵,它通过fit_transform函数计算各个词语出现的次数。通过toarray()可看到词频矩阵的结果。代码如下所示：

```
cv_fit = vector.fit_transform(train_sample_unt)
print("文本的关键词:\n", vector.get_feature_names())
print("词频矩阵:\n", cv_fit.toarray())
print("cv_fit:\n", cv_fit)
```

词频矩阵如下所示：

```
文本的关键词:
 ['00', '000', '01', '02'
词频矩阵:
 [[0 3 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 1 0]
 ...
 [0 2 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]]
cv_fit:
   (0, 3572)    14
  (0, 222)  1
  (0, 1475) 1
  (0, 3696) 1
  (0, 2639) 1
```

图1.词频矩阵

- **TfidfTransformer使用总结**

  TfidfTransformer用于统计vectorizer中每个词语的TF-IDF值。数据中的每个词语可以看作一个特征，但有很多词语的特征性并不普遍，我们认为这种单词并不能称为特征，因此我们要计算每个词语的TF-IDF值，找到每个文本中特有的词语。

  代码如下所示:

```
tfidf = tfid.fit_transform(cv_fit)
print (tfidf.toarray())
```

  矩阵如下所示:

```
[[0.         0.0531024  0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.02365076 0.        ]
 ...
 [0.         0.05408672 0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.01627009 0.         ... 0.         0.         0.        ]]
```

图2.特征矩阵

- **MultinomialNB使用总结**

  MultinomialNB假设特征的先验概率为多项式分布，即如下式:

$$P(X_j = x_{jl}|Y = C_k) = \frac{x_{jl} + \lambda}{m_k + n\lambda}$$

  其中，$P(X_j = x_{jl}|Y = C_k)$是第k个类别的第j维特征的第l个个取值条件概率。m_k是训练集中输出为第k类的样本个数。λ 为一个大于0的常数，常常取为1，即拉普拉斯平滑。也可以取其他值。

fit_prior	class_prior	最终先验概率
false	填或者不填没有意义	$P(Y = Ck) = 1/kP(Y = Ck) = 1/k$

fit_prior	class_prior	最终先验概率
true	不填	$P(Y=Ck)=mk/mP(Y=Ck)=mk/m$
true	填	$P(Y=Ck)=P(Y=Ck)=class_prior$

实验所使用的数据太过极端，pos和neg的比例达到了10：1,因此我们要不断使用二分法调试alpha参数，最终将其设置为一个极小的数字。我们最终将alpha值调到0.01,class_prior调到[0.4,0.6]得到了最优解。如下代码所示：

```
clf = MultinomialNB(alpha=0.0100, fit_prior=True, class_prior=[0.4, 0.6])
```

在做完上述工作之后，我们就可以带入数据进行预测了，代码如下所示：

```
X_train_vector = vector.fit_transform(train_sample_unt)
X_test_vector = vector.transform(test_sample_unt)
X_train = tfid.fit_transform(X_train_vector)
X_test = tfid.transform(X_test_vector)
clf.fit(X_train, train_label_unt)
test_label = clf.predict(X_test)
```

模型优化

在实际测试中我们发现，测试的结果正确率并不高，因此我们要对参数进行调节，优化准确率。我们主要对以下参数进行调节：

- CountVectorizer函数的max_features参数

max_features参数默认为None，可设为int，对所有关键词的term frequency进行降序排序，只取前max_features个作为关键词集，我们从1000开始调试，直到10000,每次增加1000，运用for循环批量获取10组数据进行测试，直到找到最佳关键词量为4000。

- MultinomialNB函数的alpha参数和class_prior参数

MultinomialNB有3个参数。其中，参数alpha即为上面的常数λ。如果发现拟合的不好，需要调优时，可以选择稍大于1或者稍小于1的数。布尔参数fit_prior表示是否要考虑先验概率，如果是false,则所有的样本类别输出都有相同的类别先验概率。否则可以自己用第三个参数class_prior输入先验概率，或者不输入第三个参数class_prior让MultinomialNB自己从训练集样本来计算先验概率，此时的先验概率为 $P(Y=Ck)=mk/mP(Y=Ck)=mk/m$ 。其中m为训练集样本总数量，m_k为输出为第k类别的训练集样本数。我们运用二分法调节alpha参数，直到得到最佳结果0.01。

三、实验环境与平台

- **操作系统：** Manjaro Linux
- **处理器：** 12 × Intel® Core™ i7-9750H CPU @ 2.60GHz
- **内存：** 15.4 GiB 内存
- **IDE：** Pycharm
- **Python版本：** 3.9
- **测试平台：** Educoder

四、实验结果与分析

实验结果如下所示：

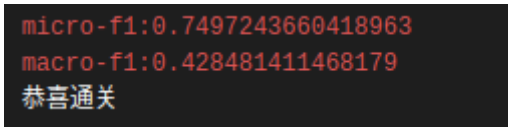


图3.Keras模型预测结果

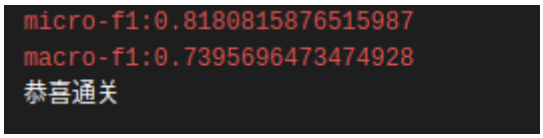


图4.Naïve Bayes模型预测结果

从上述结果我们可以看出，神经网络的准确率非常差，我推测的原因是在**本数据集上过拟合了正向样本**，我们来对其进行验证。我们用训练好的模型在训练集上跑一遍，得到的结果如下：

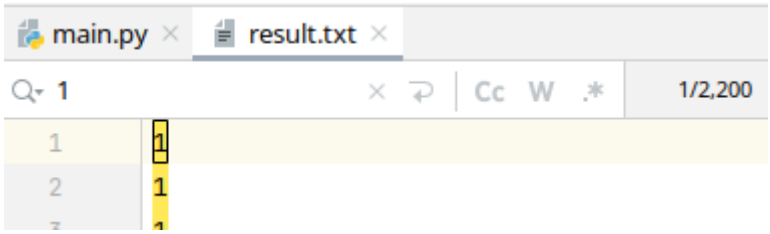


图5.epoche=5预测结果

可见测试的2200个文本label都为1,由此可见是**过拟合了正向样本**，此时epoche参数为5。

我们再将epoche设置为3,得到的结果如下：

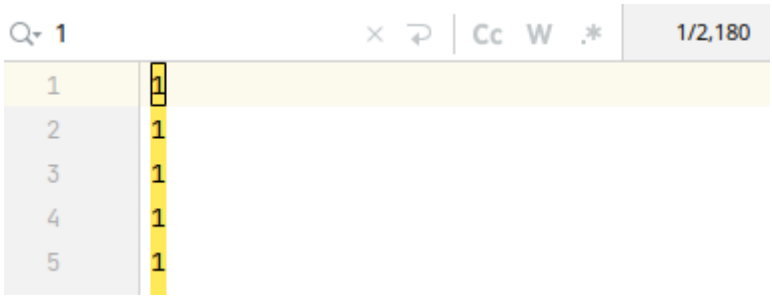


图6.epoche=3预测结果

上图说明依然是一个过拟合的状态。

神经网络的权重因为训练集中正向样本的比例过大导致的偏向于对所有的预测为正向来降低损失函数，也就是说该样本不适合运用神经网络训练模型。

而调参后的朴素贝叶斯算法，macro 达到 0.74,micro 达到了 0.81,说明可以很好的预测文本的label。

五、结论与感想

5.1 全文总结

本次实验中，我主要进行了以下的工作：

- 学习了Naïve Bayes和Keras算法相关知识，并进行了实际操作。
- 根据Naïve Bayes和Keras构建了模型并对新闻数据进行了二分类运算。
- 调节参数，优化模型。

5.2 本次课题的感想

机器学习，我自己的理解就是让计算机去模拟人类的行为方式，也就是人工智能的一个雏形，人工智能现在是一个很热门的领域。但我在本学期的学习中发现自身对机器学习乃至人工智能并没有很大兴趣，相比于训练模型和调参，我还是更喜欢一些后端和偏底层架构的工作。这也算是本次课程对我职业规划的意义。另外本次课题中老师给出的数据集不是非常完备，神经网络不能很好的完成模型训练和测试，算是一个小小的遗憾吧。

参考文献

[1] CSDN博主 curious_girl 神经网络测试结果差怎么办?

[2] CSDN博主 九点澡堂子 sklearn——CountVectorizer详解

[3] CSDN博主 CongyingWang python函数——Keras分词器Tokenizer