实验 10 语言模型与文本分类

171860607
白晋斌

Table of Contents

一、任务回顾

文本分类是为一个句子或文档分配合适类别的任务。此次任务是根据新闻的的内容来分类新闻的主题。采用的数据集是 AG's News Topic Classification Dataset 。

数据集

类别： Word, Sports, Business, Sci/Tech

训练集： train_texts.txt 每行对应一则新闻文本，train_labels.txt 每行则是对应新闻的主题； 每个类别 30000 个样本，共计 120000 个样本。

测试集： test_texts.txt 每行对应一则新闻文本，test_labels.txt 每行则是对应新闻的主题； 每个类别 1900 个样本，共计 7600 个样本。

评价指标

测试集上的错误率 err = wrong_nums/ all_nums∗ 100%

二、基于朴素贝叶斯的文本分类

读取文本

```
traintexts=pd.read_csv('train_texts.txt',header=None)
trainlabels=pd.read_csv('train_labels.txt',header=None)
testtexts=pd.read_csv('test_texts.txt',header=None)
testlabels=pd.read_csv('test_labels.txt',header=None)
traintext = traintexts[0].values.tolist()
trainlabel = trainlabels[0].values.tolist()
testtext = testtexts[0].values.tolist()
testlabel = testlabels[0].values.tolist()
text=traintext+testtext
trainlabel = list(map(lambda x: labeldic[x], trainlabel))
testlabel = list(map(lambda x: labeldic[x], testlabel))
label=trainlabel+testlabel
```

数据处理，将整个 text 集合用 tf-idf 值填充，让整个文档集成为一个 tf-idf 矩阵。这里需要使用 sklearn 的 CountVectorizer 与 TfidfTransformer 函数实现。代码如下：

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
count_v0 = CountVectorizer()
counts_all = count_v0.fit_transform(text)
count_v1 = CountVectorizer(vocabulary=count_v0.vocabulary_)
```

```
counts_train = count_v1.fit_transform(traintext)
print("the shape of train is " + repr(counts_train.shape))
count_v2 = CountVectorizer(vocabulary=count_v0.vocabulary_)
counts_test = count_v2.fit_transform(testtext)
print("the shape of test is " + repr(counts_test.shape))
```

这里有一个需要注意的地方，由于训练集和测试集分开提取特征会导致两者的特征空间不同，所以这里先用所有文档共同提取特征（counts_v0），然后利用得到的词典（counts_v0.vocabulary_）再分别给训练集和测试集提取特征。然后开始训练与测试。

```
tfidftransformer = TfidfTransformer()
train_data = tfidftransformer.fit(counts_train).transform(counts_train)
test_data = tfidftransformer.fit(counts_test).transform(counts_test)
```

最后利用朴素贝叶斯进行分类：

```
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
clf = MultinomialNB(alpha=0.01)
clf.fit(train_data, trainlabel)
preds = clf.predict(test_data)
num = 0
preds = preds.tolist()
for i, pred in enumerate(preds):
    if int(pred) == int(testlabel[i]):
        num += 1
print('precision_score:' + str(float(num) / len(preds)))
```

可以得到 90.4%的准确性。

```
the shape of train is (120000, 66464)
the shape of test is (7600, 66464)
precision_score:0.904342105263158
```

三、基于深度学习（CNN）的文本分类

因为本人之前在一些创新项目中接触过卷积神经网络，故这里尝试用卷积神经网络实现文本分类。

先把训练与测试数据放在一起提取特征，使用 keras 的 Tokenizer 来实现，将 text 文档处理成单词索引序列，单词与序号之间的对应关系靠单词的索引表 word_index 来记录；然后将长度不足 100 的新闻用 0 填充（在前端填充），用 keras 的 pad_sequences 实现；最后将标签处理成 one-hot 向量，用 keras 的 to_categorical 实现。

```
traintexts=pd.read_csv('train_texts.txt',header=None)
trainlabels=pd.read_csv('train_labels.txt',header=None)
testtexts=pd.read_csv('test_texts.txt',header=None)
testlabels=pd.read_csv('test_labels.txt',header=None)
traintext = traintexts[0].values.tolist()
trainlabel = trainlabels[0].values.tolist()
testtext = testtexts[0].values.tolist()
testlabel = testlabels[0].values.tolist()
text=traintext+testtext
trainlabel = list(map(lambda x: labeldic[x], trainlabel))
testlabel = list(map(lambda x: labeldic[x], testlabel))
label=trainlabel+testlabel

tokenizer = Tokenizer()
tokenizer.fit_on_texts(text)
sequences = tokenizer.texts_to_sequences(text
word_index = tokenizer.word_index
print('Found %s unique tokens.' %len(word_index))
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
labels = to_categorical(np.array(label))
print('Shape of label tensor:', labels.shape)
```

然后划分训练集、评估集、测试集。

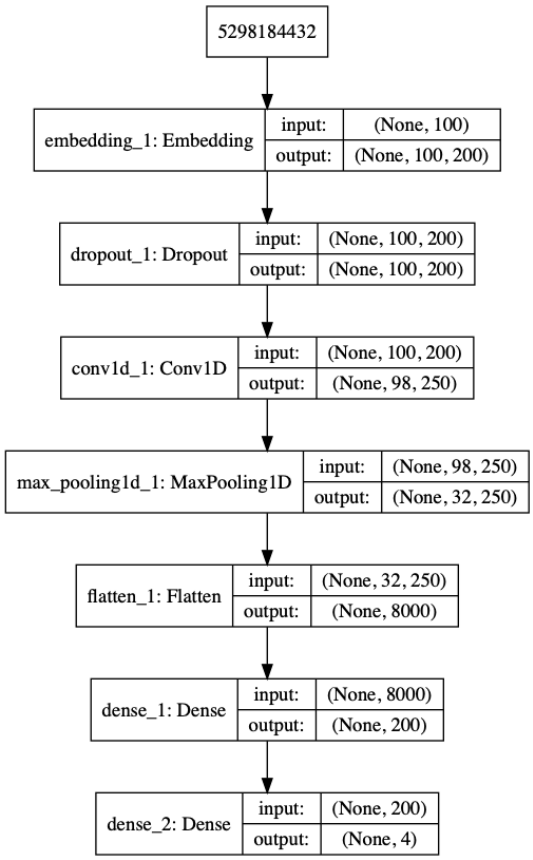
```
x_train = data[:100000]
y_train = labels[:100000]
x_val = data[100000:120000]
y_val = labels[100000:120000]
x_test = data[120000:]
y_test = labels[120000:]
```

然后就是搭建模型，首先是一个将文本处理成向量的 embedding 层，这样每个新闻文档被处理成一个二维向量，下面通过 1 层卷积层与池化层来缩小向量长度，再加一层 Flatten 层将 2 维向量压缩到 1 维，最后通过两层 Dense（全连接层）将向量长度收缩到 4 上，对应文本分类的 4 个类。

```
model = Sequential()
model.add(Embedding(len(word_index) + 1, EMBEDDING_DIM, input_length=MAX_SEQUENCE_LENGTH))
model.add(Dropout(0.2))
model.add(Conv1D(250, 3, padding='valid', activation='relu', strides=1))
model.add(MaxPooling1D(3))
model.add(Flatten())
model.add(Dense(EMBEDDING_DIM, activation='relu'))
model.add(Dense(labels.shape[1], activation='softmax'))
```

```
model.summary()
plot_model(model, to_file='model.png', show_shapes=True)
model.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['acc'])
model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=2, batch_size=128)
model.save('word_vector_cnn.h5')
print(model.evaluate(x_test, y_test))
```

模型结构如图：



最后，训练模型，测试模型。如图，在测试集达到了 92%的准确性，略高于之前的朴素贝叶斯。

```
[0.2438378861860225, 0.9197368421052632]
```

四、思考改进

在提取特征的过程中，是否可以先自行分词，之后去掉一些无意义的连词，动词之类，从而提高预测精确性？

这里的 stopwords 使用自己的一个 txt 文件。

代码如下：

```
file=open('stopwords.txt')
stopWords=file.read()
trainset=pd.read_csv('train_texts.txt',header=None)
```

```
trainlabel=pd.read_csv('train_labels.txt',header=None)
testset=pd.read_csv('test_texts.txt',header=None)
testlabel=pd.read_csv('test_labels.txt',header=None)
for i in range(7600):
    words = WordPunctTokenizer().tokenize(testset[0][i])
    #stopWords = set(stopwords.words('english'))
    wordsFiltered = []
    for w in words:
        if w not in stopWords and len(w)>2:
            wordsFiltered.append(w)
    testset[0][i]=wordsFiltered
    print(i)
testset.to_csv('test_texts2.csv', sep=',', header=False, index=False)
for i in range(120000):
    words = WordPunctTokenizer().tokenize(trainset[0][i])
    #stopWords = set(stopwords.words('english'))
    wordsFiltered = []
    for w in words:
        if w not in stopWords and len(w)>2:
            wordsFiltered.append(w)
    trainset[0][i]=wordsFiltered
    print(i)
trainset.to_csv('train_texts2.csv', sep=',', header=False, index=False)
print('write finished')
```

之后执行朴素贝叶斯和 CNN 两种文本分类方案。

朴素贝叶斯准确性 90.3%，略微下降，可以认为没有变化。

```
the shape of train is (120000, 65508)
the shape of test is (7600, 65508)
precision_score:0.9030263157894737
```

CNN 准确性 91.1%，也是略微下降。

```
[0.26624944083000485, 0.9113157894736842]
```

这说明我们去掉停用词后对实验结果产生了负面影响，初步猜测可能是自己的停用词库不够合理。自己的停用词库亦打包作为代码材料提交。

总的来说，文本分类不需要去停用词即可达到比较好的分类效果，若去停用词，则要考虑寻找好的停用词素材。