数据仓库与数据挖掘
期末作业报告

组　　员：吕文秀
专　　业：数据科学与大数据技术
提交日期：**2021.7.2**

1 分析数据集概述

（一）数据来源

公开数据集：*labeledTrainData.tsv*

说明：IMDB 影评

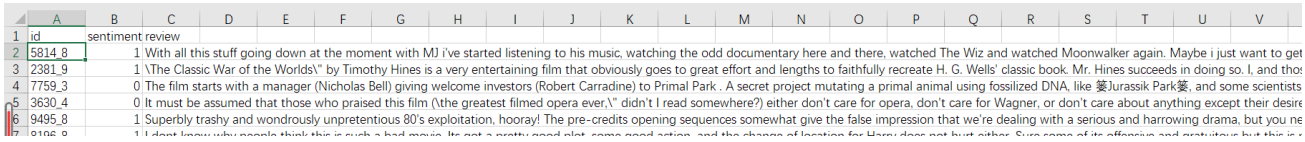
（二）属性名称与属性类型

属性名称	类型	说明
id	string	标识符
sentiment	string	情感评价，包括正面和负面
review	string	相关语句

（三）数据规模

250000 条有标签数据

（四）数据样例



	A	B	C
1	id	sentiment	review
2	5814_8	1	With all this stuff going down at the moment with MJ i've started listening to his music, watching the odd documentary here and there, watched The Wiz and watched Moonwalker again. Maybe i just want to get
3	2381_9	1	\The Classic War of the Worlds\" by Timothy Hines is a very entertaining film that obviously goes to great effort and lengths to faithfully recreate H. G. Wells' classic book. Mr. Hines succeeds in doing so. I, and thos
4	7759_3	0	The film starts with a manager (Nicholas Bell) giving welcome investors (Robert Carradine) to Primal Park . A secret project mutating a primal animal using fossilized DNA, like 篓Jurassik Park篓, and some scientists
5	3630_4	0	It must be assumed that those who praised this film (\the greatest filmed opera ever,\" didn't I read somewhere?) either don't care for opera, don't care for Wagner, or don't care about anything except their desire
6	9495_8	1	Superbly trashy and wondrously unpretentious 80's exploitation, hooray! The pre-credits opening sequences somewhat give the false impression that we're dealing with a serious and harrowing drama, but you ne

2 分析目标

挖掘目标：进行文本二分类算法实现。根据 review 对每条数据进行情感分类，为 0 为负面情感，为 1 为正面情感。

分别用词袋预处理技术、TF-IDF 预处理技术、word2vec 词向量进行文本或单词向量化；用贝叶斯分类算法和 LSTM 深度学习算法来进行分类算法。比较不同方法下的分类算法的效果。效果评价用召回率、F1 及 ROC 曲线。

3 设计

（1） 预处理：

删除标签，比如 html 标签；

删除特殊字符：非字母数字字符的特殊字符和符号通常会增加非结构化文本中的额外噪音。可以使用简单正则表达式(regexes)来实现这一点；

词根提取：词干通常是可能的单词的基本形式，可以通过在词干上附加词缀，如前缀和后缀来创建新单词。在词根提取中，我们去掉词缀以得到单词的基本形式；

删除停止词：在从文本中构造有意义的特征时，意义不大或者没有意义的词被称为停止词或停止词；

乱序分割数据集：训练集：测试集=15000：10000

（2） 特征提取：

1）词袋法：不考虑词语出现的顺序，每个出现过的词汇单独作为一列特征，只考虑词汇在文本中出现的频率。

2）Tfidf：除了考量某词汇在文本出现的频率，还关注包含这个词汇的所有文本的数量，能够削减高频没有意义的词汇出现带来的影响, 挖掘更有意义的特征。

3）词嵌入（与神经网络模型联合学习的词嵌入）：将原来稀疏的巨大的维度压缩嵌入到一个小的维度空间上，embedding 层和 word2vec 的作用和效果一样，但是 embedding 层是根据我们的任务所定，训练与我们任务有关系的词向量，和训练的任务有很大的关系，任务通常会将含义相似的词赋予取值相近的词向量值，使得网络可以更为容易的抓住相似单词之间的共性；word2vec 仅仅是使用语言模型训练出来的词向量，表示的是一个词的向量空间，使用 Word2vec 的话，往往和我们的任务有很大的距离。

（3） 训练：

生成分类器：1）经典机器学习算法：贝叶斯 2）深度学习算法：LSTM + pytorch

（4） 测试：

对建模结果进行 F1、召回率、ROC 曲线评估。

准确率（accuracy）是分类器预测正确性的比例，但是并不能分辨出假阳性错误

和假阴性错误。

召回率（recall）：召回率是覆盖面的度量，度量有多个正例被分为正例。

F1：P 和 R 指标有时候会出现的矛盾的情况，F1 综合考虑。

ROC 曲线：当测试集中的正负样本的分布变化的时候，ROC 曲线能够保持不变。

4.实现

数据预处理：处理 review 属性

（1）预处理（document_preprocess.py）：

1）

```
# 文本预处理1，字符串文本，返回list格式的单词
def text_pre_process1(text):
    text_1 = re.sub(r"</?[^>]*>|\\|n*'[\w]*|[^(\w|\s)]", ' ', text)    # 去除html标签，'\'，英文缩写，非英文字符
    text_2 = nltk.word_tokenize(text_1)    # nltk分词
    text_3 = []
    for word in text_2:
        text_3.append(stemmer.stem(word))    # nltk.stem词干提取
    return text_3
```

使用正则表达式删除 html 标签；使用 nltk.word_tokenize()进行文本分词；基于 Porter 词干提取来抽取词的词干或词根形式；

2）

```
# 获取停用词，文件为stopfile
stop_words_t = pd.read_csv(stopfile, index_col=False, sep='\t', names=['stopword'], quoting=3, encoding='utf-8')
stop_words = []
for line in stop_words_t['stopword']:
    stop_words.append(stemmer.stem(line))

# 文本处理2，处理停用词，单词list，停用词list
def stop_words_process2(word_list, stop_list):
    word_clean = []
    for word in word_list:
        if word.lower() in stop_list:
            continue
        word_clean.append(word)
    return word_clean
```

使用了 *stopfile* 文件中的停止词，并进行与 review 相同的词干提取，删除停止词。

3）

```
# 创建文件并存储list对象
contentFile_All = 'contentFile_All1.dat'
f = open(contentFile_All, 'wb')
pickle.dump(content_list, f)
f.close()
```

将 review 预处理后的结果存储到 *contentfile_all1.dat* 中，便于不同的模型

使用。

4）

```
# 数据文件
filename = "labeledTrainData.tsv"

# 创建文件并存储list对象
contentFile_All = 'contentFile_All1.dat'
f = open(contentFile_All, 'rb')
content_all = pickle.load(f)

document_info = pd.read_csv(filename,sep='\t',encoding='UTF-8')
label = document_info['sentiment'].values.tolist()
review = content_all

df_train = pd.DataFrame({'review': review, 'label': label})

# 模型选择
x_train, x_test, y_train, y_test = train_test_split(df_train['review'].values,
                                                    df_train['label'].values, train_size=15000,
                                                    test_size=10000, shuffle=False, random_state=1)
```

划分数据集得到顺序子集。

（2）特征提取：

1）bys_15k_10k_bow.py

```
#bow
cv= CountVectorizer(max_features=20000,lowercase= False)
cv.fit(words)
# 生成分类器
```

对训练集和测试集使用词袋法进行特征提取。

2）bys_15k_10k_tfidf.py

```
# 特征提取
vectorizer = TfidfVectorizer(analyzer='word', max_features=20000, lowercase= False)
vectorizer.fit(words)
```

对训练集和测试集使用 tfidf 法进行特征提取。按词频排序，仅考虑 20000 个词汇构建词汇表。

3）lstm_2.py

```
class Model(torch.nn.Module):
    def __init__(self, embedding_dim, hidden_dim):
        super(Model, self).__init__()
        self.hidden_dim = hidden_dim
        self.embeddings = nn.Embedding(vocabLimit + 1, embedding_dim)
```

与 lstm 模型联合学习的词嵌入：torch.nn 包下的 Embedding 作为训练的一层，

随模型训练得到合适的词向量。

根据预处理后的 review 创建词汇表：限制词汇表的大小：vocabLimit = 20000

```
vocabLimit = 20000
max_sequence_len = 500
obj1 = wordIndex()
for i,lines in enumerate(review):
    obj1.add_text(lines)
limitDict(vocabLimit, obj1)

class wordIndex(object):
    def __init__(self):
        self.count = 0
        self.word_to_idx = {}
        self.word_count = {}

    def add_word(self, word):
        if not word in self.word_to_idx:
            self.word_to_idx[word] = self.count
            self.word_count[word] = 1
            self.count += 1
        else:
            self.word_count[word] += 1

    def add_text(self, text):
        for i,word in enumerate(text):
            # print(word)
            self.add_word(word)

def limitDict(limit, classObj):
    dict1 = sorted(classObj.word_count.items(), key=lambda t: t[1], reverse=True)
    count = 0
    for x, y in dict1:
        if count >= limit - 1:
            classObj.word_to_idx[x] = limit
        else:
            classObj.word_to_idx[x] = count

        count += 1
```

（3）建模：

1）贝叶斯分类算法：

sklearn.naive_bayes.MultinomialNB()函数：先验为多项式分布的朴素贝叶斯。特征变量是离散变量，符合多项分布，在文档分类中特征变量体现在一个单词出现的次数，或者是单词的 TF-IDF 值等。不支持负数，所以输入变量特征的时候，别用 StandardScaler 进行标准化数据，可以使用 MinMaxScaler 进行归一化。这个模型假设特征复合多项式分布，是一种非常典型的文本分类模型，模型内部带有平滑参数。

```
# 生成分类器
classifier = MultinomialNB()
classifier.fit(cv.transform(words), y_train)

print('训练集上的预测结果：')
y_pred = classifier.predict(cv.transform(words))[0:15000]
```

2）LSTM

循环神经网络是一种用于处理序列数据的神经网络。相比一般的神经网络来说，能够处理序列变化的数据。长短期记忆（LSTM）是一种特殊的 RNN，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。相比普通的 RNN，LSTM 能够在更长的序列中有更好的表现。LSTM 的内部结构。通过门控状态来控制传输状态，记住需要长时间记忆的，忘记不重要的信息；而不像普通的 RNN 那样只仅有一种记忆叠加方式。对很多需要“长期记忆”的任务来说，尤其好用。

LSTM 模型：

LSTM 以 word_embeddings 作为输入（限制最大维度为 500），输出维度为 hidden_dim 的隐藏状态值，线性层将隐藏状态空间映射到标注空间，softmax 做最后的二分类。

隐藏层各个维度的含义是 (num_layers, minibatch_size, hidden_dim)

损失函数是 nn.NLLLoss():在 softmax 函数的基础上，再进行一次 log 运算，此时结果有正有负，log 函数的值域是负无穷到正无穷，当 x 在 0—1 之间的时候，log(x)值在负无穷到 0 之间。nn.NLLLoss 的结果就是把 log 的输出与 Label 对应的那个值拿出来，去掉负号，再求均值。

使用 Adam 优化算法，基于训练数据迭代地更新神经网络权重。

```
class Model(torch.nn.Module):
    def __init__(self, embedding_dim, hidden_dim):
        super(Model, self).__init__()
        self.hidden_dim = hidden_dim
        self.embeddings = nn.Embedding(vocabLimit + 1, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim)
        self.linearOut = nn.Linear(hidden_dim, 2)

    def forward(self, inputs, hidden):
        x = self.embeddings(inputs).view(len(inputs), 1, -1)
        lstm_out, lstm_h = self.lstm(x, hidden)
        x = lstm_out[-1]
        x = self.linearOut(x)
        x = F.softmax(x)
        return x, lstm_h

    def init_hidden(self):
        if use_cuda:
            return (
            Variable(torch.zeros(1, 1, self.hidden_dim)).cuda(), Variable(torch.zeros(1, 1, self.hidden_dim)).cuda())
        else:
            return (Variable(torch.zeros(1, 1, self.hidden_dim)), Variable(torch.zeros(1, 1, self.hidden_dim)))
if use_cuda:
    model = Model(50, 100).cuda()
else:
    model = Model(50, 100)

loss_function = nn.NLLLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

训练过程：

每次训练一条训练集的数据：准备网络输入,将其变为词索引的 Tensor 类型数据：给每个 x_train 中的单词编码，用词表中的位置来表示每个单词，才能传入 word_embeding 得到词向量，如果单词总数超过最大长度就截断；在训练每个实例前清空梯度，此外还需要清空 LSTM 的隐状态，将其从上个实例的历史中分离出来；前向传播；计算损失和梯度值，通过调用 optimizer.step() 来更新梯度；每个 epoch 计算一次平均损失，观察训练效果。

超参数：

```
# 词汇表大小
vocabLimit = 20000
# 变长输入数据不得超过 500
max_sequence_len = 500
embedding_dim=50, hidden_dim=100
# 迭代次数
epoch = 4
```

```
for epoch in range(4):
    avg_loss = 0.0
    for i in range(15000):
        # 第一步: 请记住Pytorch会累加梯度.
        # 我们需要在训练每个实例前清空梯度
        model.zero_grad()
        # 此外还需要清空 LSTM 的隐状态,
        # 将其从上个实例的历史中分离出来.
        hidden = model.init_hidden()
        # 准备网络输入, 将其变为词索引的 Tensor 类型数据
        input_data = [obj1.word_to_idx[word] for word in x_train[i]]
        if len(input_data) > max_sequence_len:
            input_data = input_data[0:max_sequence_len]
        if use_cuda:
            input_data = Variable(torch.cuda.LongTensor(input_data))
        else:
            input_data = Variable(torch.LongTensor(input_data))
        target = y_train[i]

        if use_cuda:
            target_data = Variable(torch.cuda.LongTensor([target]))
        else:
            target_data = Variable(torch.LongTensor([target]))
        # 第三步: 前向传播
        y_pred, _ = model(input_data, hidden)
        y_pred = torch.log(y_pred)
        # 第四步: 计算损失和梯度值, 通过调用 optimizer.step() 来更新梯度
        loss = loss_function(y_pred, target_data)
        avg_loss += loss.data

        if i % 1000 == 0 or i == 1:
            print('epoch :%d iterations :%d loss :%g' % (epoch, i, loss.data))
        loss.backward()
        optimizer.step()
    torch.save(model.state_dict(), 'model__1' + str(i + 1) + '.pth')
    print('the average loss after completion of %d epochs is %g' % ((i + 1), (avg_loss / 15000)))
```

模型保存和加载：训练周期较长；方便对模型进行评估

模型保存：

```
    torch.save(model.state_dict(), 'model__1' + str(i + 1) + '.pth')
```

模型加载（model_predict.py）：

```
model.load_state_dict(torch.load("model__115000.pth"),strict=False)
```

4）模型结果：

评估方法：

```
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
```

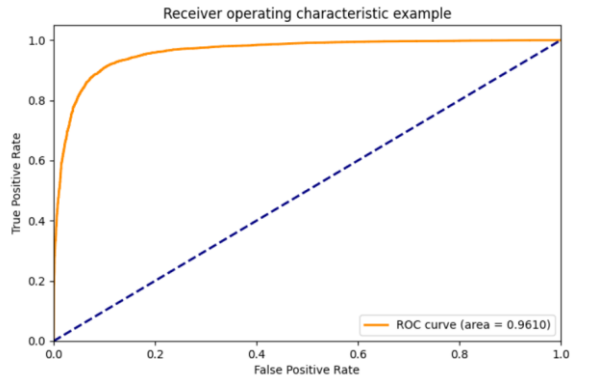
```
print('召回率为：',recall_score(y_train,y))
print('F1：',f1_score(y_train,y))
print('accuracy_score：',accuracy_score(y_train,y))
```

Bys_15k_10k_bow.py：

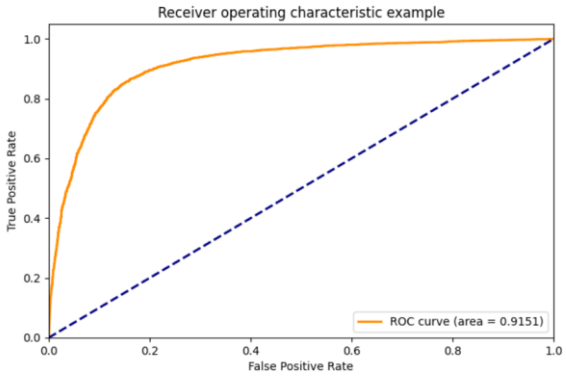
限制词表大小为 20000。

```
训练集上的预测结果：
召回率为： 0.85428075330573
F1： 0.8563395367519079
accuracy_score： 0.8569333333333333
测试集上的预测结果：
召回率为： 0.8358268501895073
F1： 0.8396793587174348
accuracy_score： 0.84
```

训练集：



测试集：

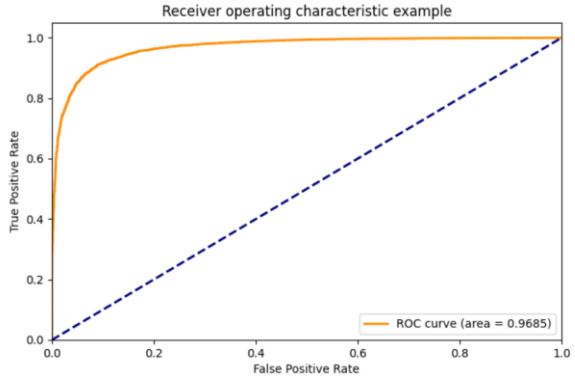


bys_15k_10k_tfidf.py：

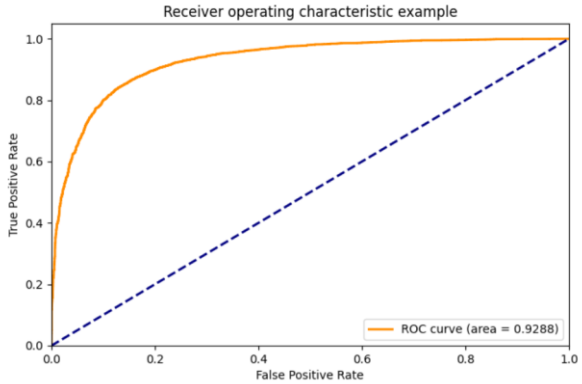
限制词表大小为 20000。

```
训练集上的预测结果：
召回率为： 0.9041004407639909
F1： 0.908651587354856
accuracy_score： 0.9092666666666667
测试集上的预测结果：
召回率为： 0.8517853580690206
F1： 0.8550260312374849
accuracy_score： 0.8552
```

训练集



测试集：



Lstm_2.py

超参数：

vocabLimit = 20000

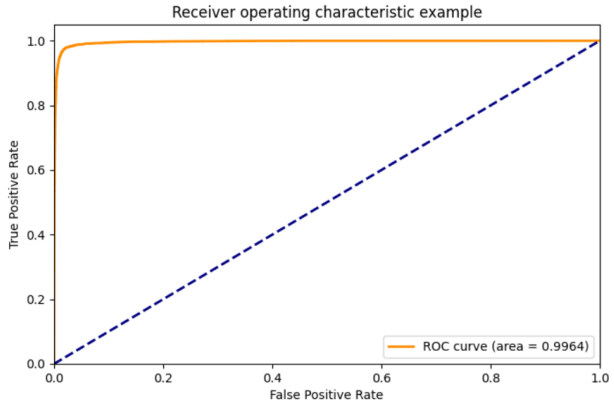
max_sequence_len = 500

embedding_dim=50, hidden_dim=100

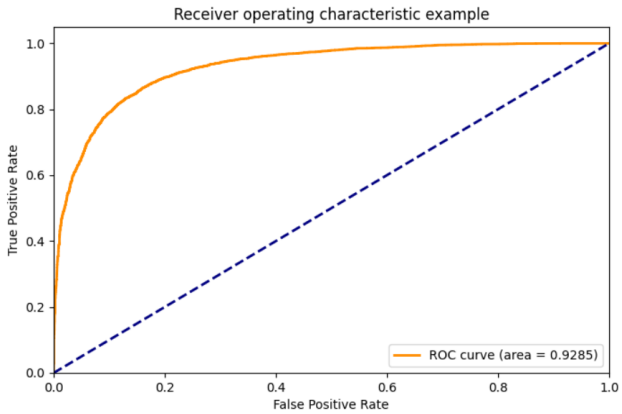
epoch = 4

```
训练集：
召回率为： 0.9465740617069587
F1： 0.9676406335335881
accuracy_score： 0.9684
D:/大三下/数据仓库与数据挖掘/Ducumer
  x = F.softmax(x)
测试集：
召回率为： 0.7863554757630161
F1： 0.8352579722428223
accuracy_score： 0.8445
```

训练集：



测试集：



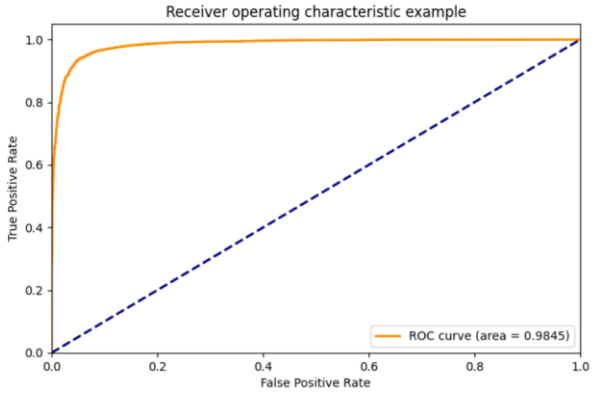
模型完美拟合训练集，但是在测试集上效果不佳，考虑是过拟合的原因。网络越大，功能越强，更容易过拟合。

1）降低迭代次数

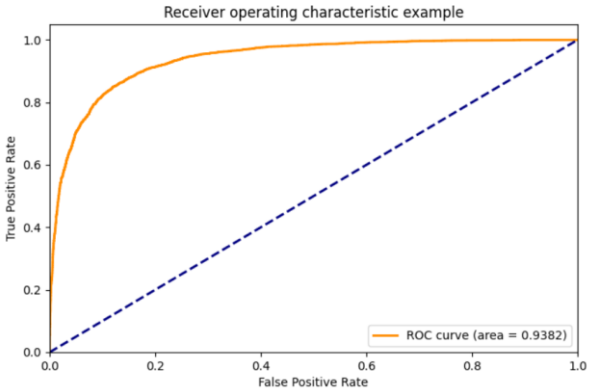
Epoch = 2

召回率为： 0.9057032189127822
F1： 0.9326731311464136
accuracy_score： 0.9347333333333333
D:/大三下/数据仓库与数据挖掘/DucumentDivi
x = F.softmax(x)
测试集：
召回率为： 0.8047077598244564
F1： 0.8494419877869025
accuracy_score： 0.857

训练集：



测试集：

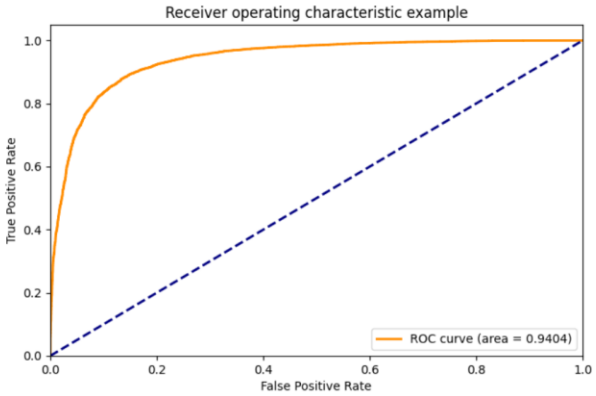


有效果：train loss 上升，val loss 下降，确定是网络过拟合，继续降低：

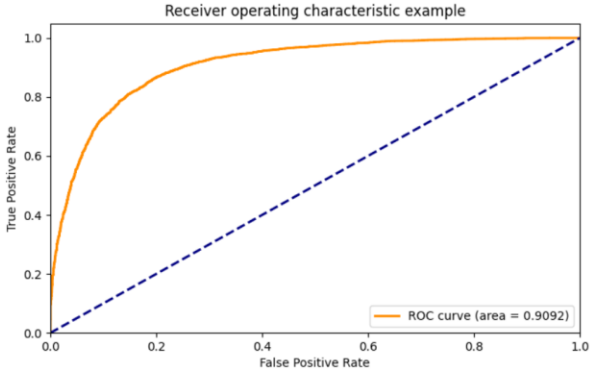
Epoch = 1

召回率为： 0.8509416321624148
F1： 0.86697965571205
accuracy_score： 0.8696666666666667
D:/大三下/数据仓库与数据挖掘/DucumentDivid
x = F.softmax(x)
召回率为： 0.8084979054458408
F1： 0.8270584634220998
accuracy_score： 0.8305

训练集：



测试集：



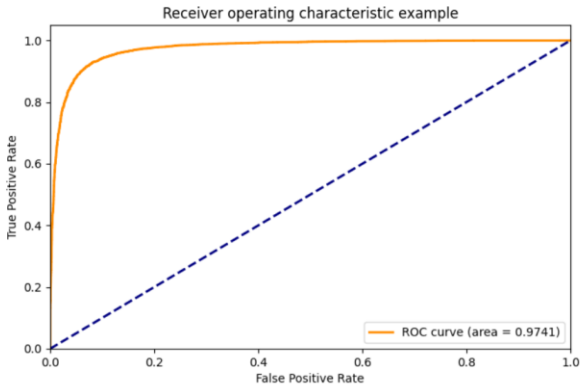
train loss 上升，val loss 上升，出现欠拟合的问题。

2）加入 dropout

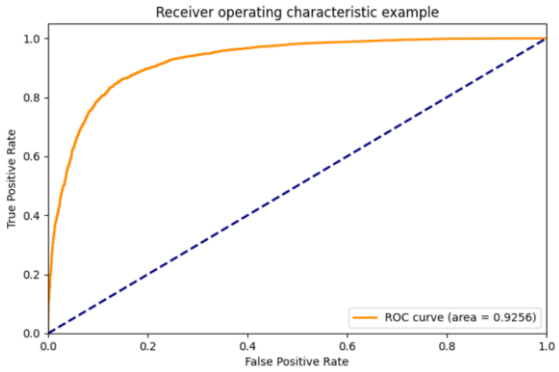
Epoch = 2 Dropout = 0.2

```
召回率为： 0.9026312274609323
F1： 0.9188931946427357
accuracy_score： 0.9204666666666667
D:/大三下/数据仓库与数据挖掘/DucumentDivide2/Duc
  x = F.softmax(x)
召回率为： 0.824655894673848
F1： 0.8480000000000001
accuracy_score： 0.8518
```

训练集：



测试集：



在提升召回率上有较大效果，但是综合效果没有提升。

5）模型评估效果分析：

		accuracy_score	recall_score	F1_score	ROC
bys_15k_10k_bow.py	训练集	0.90	0.88	0.90	0.9610
	测试集	0.85	0.83	0.85	0.9151
bys_15k_10k_tfidf.py	训练集	0.91	0.90	0.91	0.9685
	测试集	0.86	0.85	0.86	0.9288
lstm_2.py epoch=2	训练集	0.93	0.91	0.93	0.9845
	测试集	0.86	0.80	0.85	0.9382
dropout=0.2 epoch=2	训练集	0.98	0.97	0.98	0.9741
	测试集	0.85	0.82	0.85	0.9256

（1） 预处理比较：经过 tfidf 预处理后的贝叶斯分类模型在各方面的效果都比词袋预处理模型好，但是差距不大。

结论：词袋模型只考虑某词汇在文本出现的频率，Tfidf 除了考量某词汇在文本出现的频率，还关注包含这个词汇的所有文本的数量，能够削减高频没有意义的

词汇出现带来的影响，挖掘更有意义的特征。

（2） 模型比较：LSTM 在训练集上各方面的预测效果都明显好于贝叶斯分类模型。

结论：离散贝叶斯是基于词与词是相互独立的，没有联系的，LSTM 是 RNN 的改进版，一定程度上规避了 RNN 的梯度消失的问题。理论上，它可以训练到词与词之间，句子与句子之间上下文的潜在的联系。

bys_15k_10k_tfidf.py	训练集	0.91	0.90	0.91	0.9685
	测试集	0.86	0.85	0.86	0.9288
lstm_2.py epoch=2	训练集	0.93	0.91	0.93	0.9845
	测试集	0.86	0.80	0.85	0.9382

LSTM 在测试集的综合效果(AUC 值和 F1score)和贝叶斯分类相比没有太大优势，考虑到神经网络的复杂性和调参的复杂性，可能没有训练出最好的 LSTM 模型。

5 数据分析总结

1、不了解 word2vec 与神经网络的结合使用：在深度学习中作为 Embedding 层，完成从高维稀疏特征向量到低维稠密特征向量的转换。

embedding_dim 的选择要注意根据自己的符号数量，举个例子，如果词典尺寸是 1024，那么极限压缩（用二进制表示）也需要 10 维，再考虑词性之间的相关性，怎么也要在 15-20 维左右，虽然 embedding 是用来降维的，但是也要注意这种极限维度，结合实际情况，合理定义。

2、某个模型在训练集上效果很好，但是在测试集上的结果并不如意，可能的原因： (i).在训练集上过拟合；（网络越大，功能越强，但也更容易过拟合。）(ii).测试集数据比训练集数据更难区分，这时，有必要去进行模型结构，算法方面的修改；(iii).测试集并不一定更难区分，只是与训练集的分布差异比较大，那么此时如果我们去想方设法提高训练集上的性能，这些工作都将是白费努力。

当网络过拟合时，可以采用的方式是正则化（regularization）与丢弃法（dropout）以及 BN 层（batch normalization），正则化中包括 L1 正则化与 L2 正则化，在 LSTM 中采用 L2 正则化。另外在使用 dropout 与 BN 层时，需要

主要注意训练集和测试集上的设置方式不同，例如在训练集上 dropout 设置为 0.5，在验证集和测试集上 dropout 要去除。

当网络欠拟合时，可以采用的方式是：去除/降低正则化；增加网络深度（层数）；增加神经元个数；增加训练集的数据量。

3、学习了文本分类的基本过程以及不同的预处理方法对分类效果的影响，不同的分类器尤其是循环神经网络在文本分类领域的应用，明白了 Word2vec 的出现对文本分类的重要影响。文本分类是 NLP 领域比较经典的使用场景；文本分类一般分为：特征工程+分类器+结果评价与反馈。特征工程分为：文本预处理+特征提取+文本表示。

6 分工说明

吕文秀

7 参考文献

[1] Kamekin.LSTM 调参经验. https://www.cnblogs.com/kamekin/p/10163743.html.2018/12/23