华北水利水电大学

自然语言处理

实验报告

20 22 ——20 23 学年

第 一 学期

实验报告序号：实验报告（四）

实验名称：基于 LSTM 的情感分析实现

学生专业班级：人工智能 2020185

学生姓名：高树林

学　　号：202018526

专　　业：人工智能

一、实验目的

1.掌握情感分类的基本原理。

2.掌握采用 lstm 进行情感分类的基本方法。

3.理解数据集大小对深度学习模型表现的影响。

4.理解训练集类别不平衡问题对模型表现的影响。

二、实验内容

基于给定“pos.txt”和“neg.txt”两个 txt 文件，划分训练集和数据集，基于训练集训练情感分类器，并在测试集上进行准确率的测试。

三、实验要求

(1)比较不同数据量下，利用 lstm 模型进行情感分类的准确率，并对结果进行分析。数据量设定分为 3 种：从“pos.txt”和“neg.txt”中分别选择 20%的数据量进行情感分析任务；分别选择 50%的数据量进行情感分析任务；分别选择 100%的数据量进行情感分析任务

(2)比较训练集中“积极”和“消极”两个类别的数据量比重不同的条件下利用 lstm 模型进行情感分类的准确率表现，并对结果进行分析。训练集中的数据量比重设置分为 3 种：“积极”数据 860 条，“消极”数据 1900 条，剩余用于测试；“积极”数据 500 条，“消极”数据 1900 条，剩余用于测试；“积极”数据 100 条，“消极”数据 1900 条，剩余用于测试。

四、文本分类

4.1 采用 LSTM 进行情感分类的实现流程分析。

补充内容

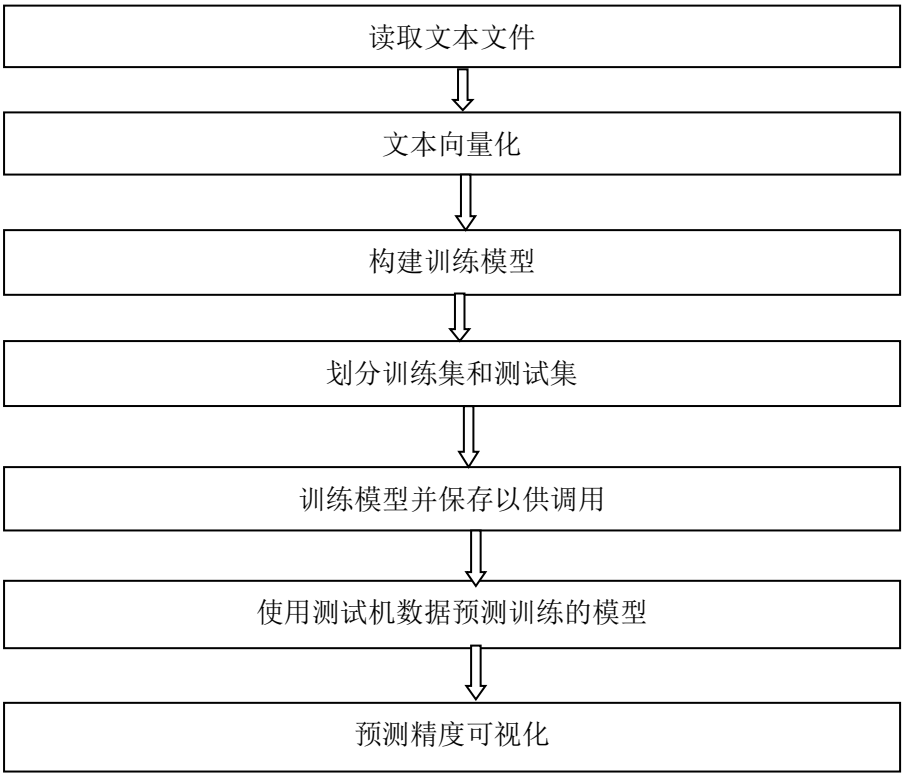

图 1 情感分类流程图

4.2 情感分类的实现步骤、关键代码及解释分析。

补充内容

步骤 1：读取文本文件。本次实验的文本格式为.txt 文本格式，读取时按行读取并并实现对评论文本的分词操作，并把分词操作的结果加到列表中，同时要加上该文本的标签。其实现的代码如下所示：

```
1. f_pos = open('pos.txt', 'r', encoding='utf8')
2. for line in f_pos:
3.     if line.strip('\n') is not None:
4.         tokens = list(jieba.cut(line))
5.         processed_sent = " ".join(tokens)
6.         review_data.append(processed_sent)
7.         labels.append(1)
```

在上述代码中，首先按行读取每一行，并利用 jieba.cut()函数实现文本分词，用空格将每一个分词连接起来后将其加到 review_data 列表里面构成数据列表，同时在对应的标签列表 labels 后面加上该文本所对应的标签。在这里 1 表示好评，0 表示差评。

步骤 2：实现文本向量化。文本向量化的目的是将文本特征用数据来表示，以便于及其识别与学习。本实验采取 Tokenizer 类来实现文本的向量化，其主要的核心代码和解释如下：

```
1. tokenizer = Tokenizer() # 创建一个 Tokenizer 对象，将一个词转换为正整数
2. tokenizer.fit_on_texts(review_data) #将词编号，词频越大，编号越小
3. word2index = tokenizer.word_index
4. vocab_size=len(word2index)
5. #print(vocab,len(vocab))
6. index2word = {word2index[word]:word for word in word2index}
7. x_word_ids = tokenizer.texts_to_sequences(review_data) #将句子中的每个词转换为数字
8. print(x_word_ids[1])
9.
10. x_padded_seqs = pad_sequences(x_word_ids,truncating='post',maxlen=100)#将每个句子设置为
11. x_padded_seqs=np.array(x_padded_seqs)
12. print(x_padded_seqs[1])
```

在上述代码中，首先创建了 tokenizer 对象，其 fit_on_text()函数读取整个列表，将分词全部编号，这里的编号特征是词频越大、编号越小。第 6 行实现的功能是交换字典键和值的值，以编号为键，分词为值。经过第 7 行操作，所有的分词能转换为数字，最后经 pad_sequenxe()函数处理得到一维列表，一维列表经过重组成为数组类型，将分词映射为 100 维后最终实现文本的向量化。

步骤 3：构建模型。本实验构建的是具有 Embedding、LSTM、Softmax 的网络结构。其代码如下所示：

```
1. # 创建深度学习模型， Embedding + LSTM + Softmax.
2. def create_LSTM(n_units, input_size, output_dim,vocab_size):
3.     model = Sequential()
4.     model.add(Embedding(input_dim=vocab_size + 1, output_dim=output_dim,
5.     input_length=input_size, mask_zero=True))
6.     model.add(LSTM(n_units, input_shape=(None,input_size)))
7.     model.add(Dropout(0.2))
8.     model.add(Dense(output_dim , activation='softmax'))
9.     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
10.      model.summary()
11.      return model
```

在上述代码中，函数主题第一行定义了 model 网络，之后可以在 model 上进行修改和添加网络结构。接着就是在 model 网络上添加了一个嵌入层，之后又添加了一个 LSTM 层，接着是加一层 Dropout 层，加这层网络的原因是 Dropout 网络能够将根据概率，删除某一部分的神经元，不仅能够使网络更加轻量化，更重要的是它能够防止过拟合现象的发生。之后又在其上增加了一层 desen 层，其作用就是由一个特征空间线性变换到另一个特征空间。本质原理就是通过卷积操作，将前面提取的特征，在 dense 经过非线性变化，提取这些特征之间的关联，最后映射到输出空间上。最后选择损失函

数和优化器进行反向传播和跌代。

步骤 4：划分数据集。训练集和测试集相同的可以通过 K 折交叉验证的方法来得到划分后的测试集和训练集；训练集和测试集不相同的可以通过列表的切片操作得到划分后的训练集和测试集。其核心代码和解释如下示：

```
def split_data(x_padded_seqs,labels,K):  # 划分数据集和测试集
    c = list(zip(x_padded_seqs,labels))
    shuffle(c)
    x_padded_seqs,labels, = zip(*c)  # 打乱顺序
    if 0 < K <= 1:
        x_padded_seqs, labels = x_padded_seqs[0:int(len(x_padded_seqs)*K)],labels[0:int(len(x_padded_seqs)*K)]
        return train_test_split(x_padded_seqs,labels,test_size=0.2,shuffle=0)
    else:
        train = list(x_padded_seqs)
         labels = list(labels)  # 打乱结果转换成列表
         return train[0:K]+train[-1900:],train[K:-1900],labels[0:K]+labels[-1900:],labels[K:-1900]
```

在上述代码中，传入的参数分别是词向量数据和标签数据还有 K 值。K 值是最重要的，它决定了是利用五折交叉分割数据集还是利用列表切片的形式分割数据集。其思想为：当传入的 K 值在 0 到 1 之间，认为它是利用五折交叉验证的方法来分割数据集，就利用 train_test_split()函数进行分割。K 值不为 0-1 的数，就认为它是要取的 POS 类别的词向量个数。当然，在这之前一定要先将词向量和标签集同步打乱。

步骤 5：训练并保存模型。网络和数据集构建完毕之后，需要将数据集放进网络拟合迭代，得到一个最优权重。所谓模型训练就是这样的一个过程。模型训练的代码和相关解释如下所示：

```
def model_train():
    # 模型输入参数，需要自己根据需要调整
    input_size = x_padded_seqs.shape[1]
    print(input_size)
    n_units = 100
    batch_size = 32
    epochs = 200
    output_dim = 2
    # 模型训练
     lstm_model = create_LSTM(n_units, input_size, output_dim,vocab_size=vocab_size)
    lstm_model.fit(x_train, y_train_onehot, epochs=epochs, batch_size=batch_size, verbose=1)
    return lstm_model
```

在上述代码中，第 10 和第 11 行为核心代码，在其之前是求超参数的过程。Input_size 是之前的分词向量的维度，之前提到过分词最后被划分的维度维 100 维，因此这里 input_size 维 100。最后利用 fit()方法进行训练，巡检结束后将 model 作为函数的返回值，以便后续调用。

步骤 6：使用模型在验证集上求模型准确率。该步骤分为 3 步，加载模型、测试集预测。详细的代码和解释如下所示。

```
1. from sklearn.metrics import accuracy_score
2. lstm_model = model_train()
3. results=lstm_model.predict(x_test)
4. result_labels = np.argmax(results, axis=-1) # 获得最大概率对应的标签
5. #print(result_labels)
6. print('准确率', accuracy_score(y_test, result_labels))
```

在上述代码中，首先实现的是将模型训练的返回值赋给 lstm_model，用该模型预测 x_text 得到预测结果列表，在该列表中获取的概率最大值对应的预测结果就是预测值。

步骤 7：预测结果可视化。下述代码是在预测过程中的精度变化曲线。其能反映在训练整个过程中，正确率的变化。

```
1. from skPrecision = []
2. correct = 0
3. for i in range(len(result_labels)):
4.     if result_labels[i] == y_test[i]:
5.         correct += 1
6.     Precision.append(correct / (i + 1))
7. num = [i+1 for i in range(len(result_labels))]
8. plt.plot(num, Precision, c='red')
9. # plt.scatter(num,Precision, c='red')
10.  plt.grid(True, linestyle='--', alpha=0.5)
11.  plt.xlabel("累计测试样本个数", fontdict={'size': 16})
12.  plt.ylabel("实时准确率", fontdict={'size': 16})
13.  plt.title('Precision', fontdict={'size': 20})
14.  plt.savefig('Precision.jpg')
15.  plt.show()
```

在该步骤利用 for 循环获取每一次的正确率，利用列表生成式生成一个 1 到测试集长度的列表并赋值给 num。以 num 为横坐标，Precision 为纵坐标，通过 matplotlib 工具包中的 pyplot 函数的 plot 函数就能画出折线图。4、5、6、7 三行分别为设置布局、横坐标名称、纵坐标名称和标题。

五、结果及分析

5.1 不同数据量下的情感分类结果与分析

实验结果：

结果表格样例：、

ID	数据使用量	准确率
1	100%	94.25%
2	50%	90.23%
3	20%	67.51%

可视化结果 1：利用 matplotlib 工具包，对分别从两个文档选取所有数据作为训练集做情感分类得到的精度曲线如下图 1 所示。

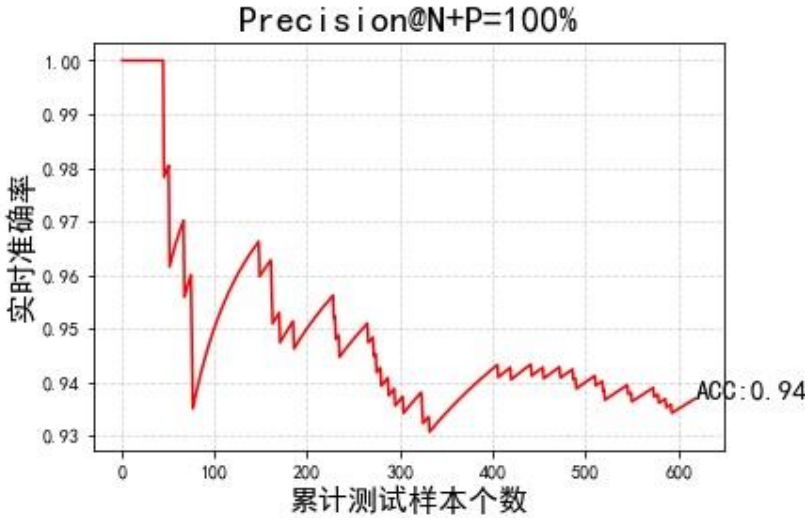

图 1 训练数据为 100%时的精度图

可视化结果 2：利用 matplotlib 工具包，对分别从两个文档选取 50%数据作为训练集做情感分类得到的精度曲线如下图 2 所示。

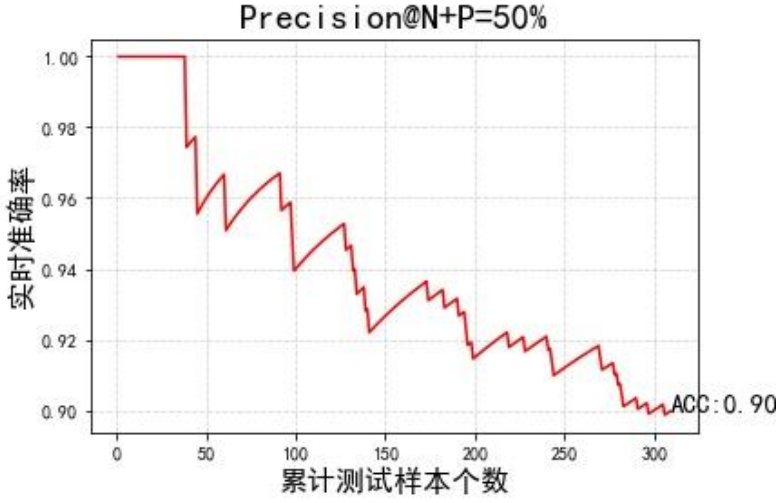

图 2 训练数据为 50%时的精度图

可视化结果 3：利用 matplotlib 工具包，对分别从两个文档选取 20%数据作为训练集做情感分类得到的精度曲线如下图 3 所示。

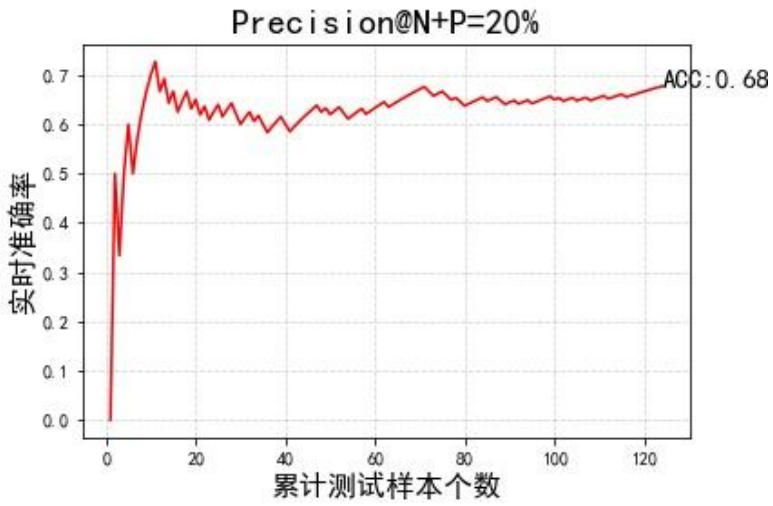

图 3 训练数据为 20%时的精度图

对各种条件下的结果进行对比分析。

上述三个实验训练集和测试集的数据总和分别约为整个数据集的 20%、50%、100%。其中，在每次实验中，测试集：训练集==1：4。按照该数据样本训练模型得到如上表所示的数据所示。结果是当只用了 20%的数据集时，得到的模型在训练集上效果达到 99%，但是在测试集上的效果只有 68%，导致这种结果的原因可能是数据量太少，模型只能“委屈自己”来适应数据，从而表现出一种过拟合的现象。随之数据的不断增多，当数据量达到 50%的时候，模型在训练集和测试集上的效果都在 90%以上；当数据量达到 100%时，模型在训练集和测试集上的效果较 50%的数据量来说，准确率提升并不是很高，这可能是因为模型已经接近拟合了，继续增加数据不会再提高其准确度了，甚至还会在一定程度上降低模型的准确度。因此一般在大型的网络架构训练模型时会隔若干轮保存一次，找到最好的模型。

不难看出，在二分类训练中，在一定范围内，训练集和测试集所占比例相当的情况下，数据越多模型拟合的效果一般会越好，但这并不是绝对的。这要结合模型网络的复杂程度分析。一般来说，太少的示例将导致较低的测试准确性，这可能是因为所选模型过于适合训练集或训练集不足以代表问题。太多的示例将导致良好的测试精度，但可能会略低于理想的测试精度，这可能是因为所选模型没有能力学习这么大的训练数据集的细微差别，或者该数据集过分地代表了问题。

5.2 类别不平衡下的情感分类结果与分析

实验结果：

结果表格样例：

ID	训练集中积极和消极的比例	准确率
1	Pos860+Neg1900	92.49%
2	Pos500+Neg1900	83.21%
3	Pos100+Neg1900	68.01%

可视化结果 4：利用 matplotlib 工具包，对分别从两个文档选取 860 条、1900 条数据作为训练集做情感分类得到的精度曲线如下图 4 所示。

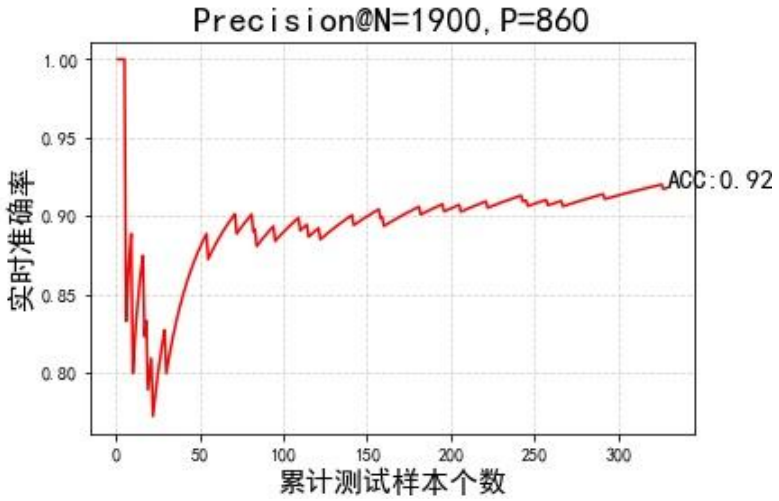

图 4 训练集为 Pos860+Neg1900 时的精度图

可视化结果 5：利用 matplotlib 工具包，对分别从两个文档选取 500 条、1900 条数据作为训练集做情感分类得到的精度曲线如下图 5 所示。

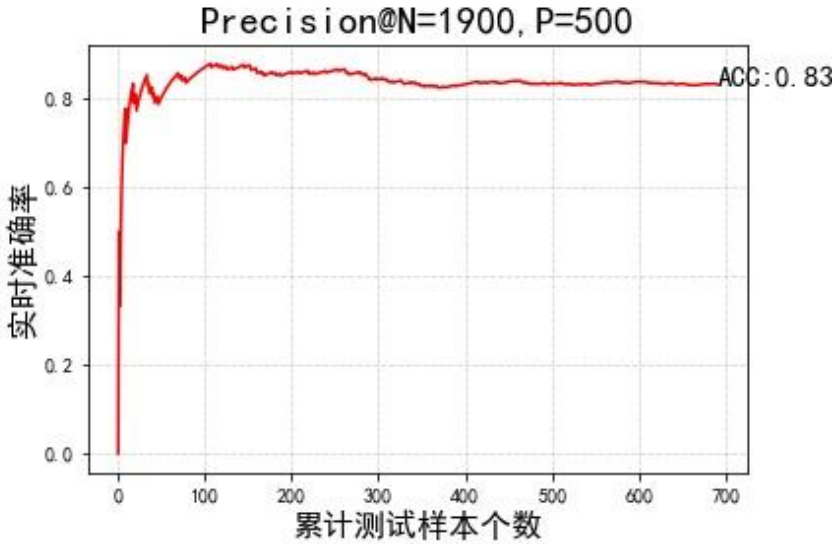

图 5 训练集为 Pos500+Neg1900 时的精度图

可视化结果 6：利用 matplotlib 工具包，对分别从两个文档选取 100 条、1900 条数据作为训练集做情感分类得到的精度曲线如下图 6 所示。

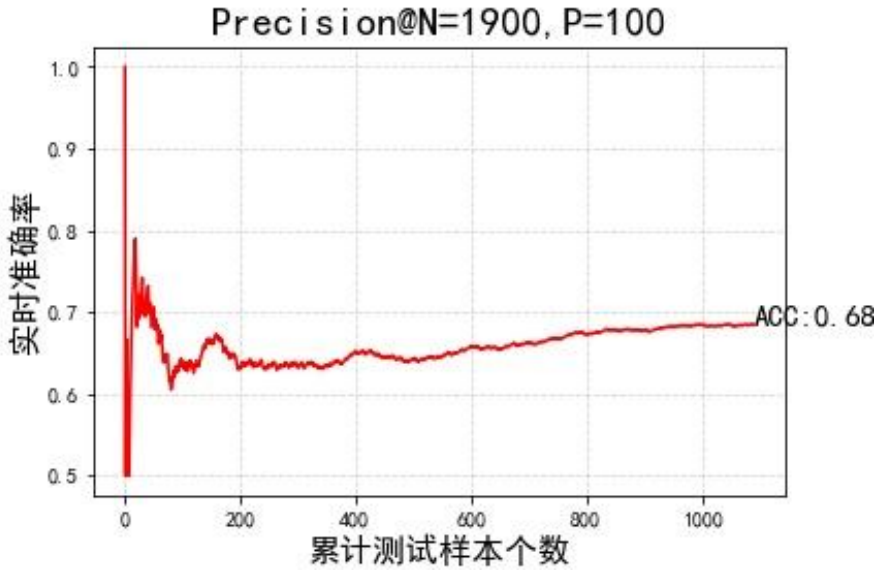

图 6 训练集为 Pos100+Neg1900 时的精度图

对各种条件下的结果进行对比分析。

上述三个实验中，训练集中 Neg 的数量始终不变，为 1900 条，Pos 的数量一直处于变化之中，分别为 100 条、500 条、860 条。实际上，这相当于多分类训练集的比例不同。其比例分别为 19：1、4：1、2.2:1。本实验是基于这三个比例下探究的训练集类别不平衡问题对模型表现的影响。实验结果是当只用了 Neg:Pos=19：1 的数据集时，得到的模型在训练集上效果达到 99%，但是在测试集上的效果只有 68%，导致这种结果的原因可能是数据不平衡，在训练过程中，模型学到的关于 Neg 的特征比 Pos 多得多，因此训练出来的模型效果并不是非常好的。因此用这样的模型在测试集上进行测试的效果并不会很好。 随之数据的不断增多，当使用 Neg=1900，Pos=860 的数据集时，模型在训练集和测试集上的效果都在 90%以上。

在数据都是正常均衡的情况下，上述实验在测试集上的效果理论上不会很好，甚至不会超过 50%。因此针对该问题我做了数据分析。

最开始我查取了在本次实验所提供的数据各类别的数据量，不出所料，数据是不平衡的，其中 Pos 类别的数据是 2142 条，Neg 类别的数据是 947 条。这样的话，在测试集上，上述三个实验所对应的测试集数据分别为 1389（Neg=342，Pos=847）,989（Neg=342,Pos=447）,629(Neg=342,Pos=87)。通过上述数据结合训练集数据得到的结果如下表 1 所示。

表 1 数据不均衡实验的数据详情

	1900：100		1900：500		1900：860	
	Neg	Pos	Neg	Pos	Neg	Pos
训练集	1900	100	1900	500	1900	860
测试集	342	847	342	447	342	87
训练集准确率	100%		100%		100%	
测试集准确率	68%		83%		92%	

通过表 1 不难看出，1900：100 的效果是最差的。这可能是因为选取的训练集的比例悬殊，造成模型出现了过拟合的现象。模型在训练的时候即使把所有数据都预测为 Neg 类别正确率也能高达 95%。好在定义的网络中添加了一层 dropout 层，保证按照一定的概率将其暂时从网络中丢弃，这样可以比较有效的缓解过拟合的发生，在一定程度上达到正则化的效果。后来随着 Pos 类别的训练数据不断增多，模型的效果变的更好了。这说明在数据分布不均衡时，数据量越大，且不同类别数据的数量越接近，模型的效果也就越好。