华东师范大学数据科学与工程学院实验报告

课程名称：人工智能	**年级**：2019	**上机实践成绩**：
指导教师：李翔	**姓名**：张雯怡	
上机实践名称：文本分类	**学号**：10195501425	**上机实践日期**：2022.2.25
上机实践编号：Project 1	**组号**：	**上机实践时间**：2022.3.16

一、实验目的

学习使用深度学习中的模型完成一个文本多分类任务。

二、实验任务

选用深度学习模型，根据已知标签(一共是 label 0 - label 9 十个类别)的文本语料(train_data.txt)对模型进行训练和评估，再使用训练后的模型去预测测试集(test.txt)上的标签，即对其进行分类，且每条文本的预测分类只能对应 label 0-9 中的一个类别。

这次实验中我使用的模型是 LSTM（长短期记忆网络）。

三、使用环境

Anaconda JupyterNotebook python 3

环境依赖及一些库函数：

numpy、pandas、nltk 用于处理数据

tensorflow 2.8.0 keras 用于模型建立：

from tensorflow.keras.preprocessing.text import Tokenizer 词条化

from tensorflow.keras.preprocessing.sequence import pad_sequences 序列化

from tensorflow.keras.models import Sequential 建模

from tensorflow.keras.layers import Dense, Embedding, LSTM, SpatialDropout1D

神经网络的嵌入层、全连接层、LSTM 层等

from tensorflow.keras.callbacks import EarlyStopping 防止过拟合

sklearn 0.0 用于评估模型：accuracy_socre、classification_report

matplotlib 用于绘图

四、实验过程

1. 文档读取

train_data.txt 格式如下：

{"label": 0, "raw": "……"}

{"label": 1, "raw": "……"}

……

{"label": 9, "raw": "……"}

将文本按行存储到列表中，用 pandas 中的 DataFrame 将字符串转化为表格的格式，

DataFrame 同时将 label 和 raw 分成表格中的两列，便于接下来模型训练：

Out[32]:

	label	raw
6659	8	I like wearing these earrings - they are simpl...
3308	4	I am 45 and this stuff works. I have used a l...
218	0	All the parts were there for a truly fine seri...
6139	7	"$6 for a battery? too good to be true?" That'...
7297	9	Classic album, many classic albums dropped aro...

拿到数据后查看数据是否存在空值或异常：

Out[33]:

	label	count
0	0	800
1	1	800
2	2	800
3	3	800
4	4	800
5	5	800
6	6	800
7	7	800
8	8	800
9	9	800

一共是 0-9 十分类，每个类别中有 800 条语料。

2. 数据预处理

由于 label 已经是 id 型的数据，可以直接在分类模型上训练不必对 label 做索引处理。

对于英文文本，删除对文本分类无用的符号、停用词，这里直接引入 nltk 中英文单词的常用停用词表进行处理。

不像中文分词那么麻烦，英文文本的只需要根据空格” ”来分词即可。

经过以上处理得到清洁和分词好的文本(cut_raw)可以直接放入模型训练：

Out[36]:

	label	raw	clean_raw	cut_raw
0	0	I only watched the Wanda Sykes portion of this...	I only watched the Wanda Sykes portion of this...	I watched Wanda Sykes portion ...
1	0	This is a cute series, and I did watch two epi...	This is a cute series and I did watch two epis...	This cute series I watch two...
2	0	This series revolves around 4 friends who run ...	This series revolves around 4 friends who run ...	This series revolves around 4 friend...
3	0	I was pleasantly surprised with this "out of t...	I was pleasantly surprised with this out of th...	I pleasantly surprised box ...
4	0	I heard about It's Always Sunny from a Kevin S...	I heard about Its Always Sunny from a Kevin Sm...	I heard Its Always Sunny Kevin...

3. LSTM 建模

数据预处理完成后，就开始进行 LSTM 的建模工作：

I. 将 cut_raw 数据进行向量化处理，把每条 cut_raw 转化成一个整数序列的向量；

II. 设置最频繁使用的 50000 个单词；

III. 填充 X，让 X 的各个列的长度统一，设置每条 cut_raw 的最大单词数目为 250 个，超过的截去，不足的补 0。（实际上 250 的长度已经足够覆盖每行文本）

IV. 拆分训练集和测试集：train_size=0.95，test_size=0.05.

V. 定义一个 LSTM 序列模型：

```
# 定义模型
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(len(label_list), activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, 300, 100)          5000000

 spatial_dropout1d_1 (Spatia  (None, 300, 100)         0
 lDropout1D)

 lstm_1 (LSTM)                (None, 100)               80400

 dense_1 (Dense)              (None, 10)                1010

=================================================================
Total params: 5,081,410
Trainable params: 5,081,410
Non-trainable params: 0
_________________________________________________________________
```

• 模型的第一层是 Embedding 嵌入层，使用长度为 100 维的向量来表示每一个单词；

Embedding Layer 其实就是 lookup table，具有降维的作用。如果使用 one-hot representation 来表示所有单词，one-hot 编码的长度即是词库大小，在庞大的词汇量情况下它的长度会非常大，并且 vector 是非常稀疏的，导致难以把握词汇之间的相似性，在语言模型的训练中相似词语应该以相似的稠密 vector 来表示才更便于利用向量之间角度来衡量词语之间的相似性。Embedding Layer 的作用就是将原始 one-hot 编码的单词映射到低维向量表示。

• SpatialDropout1D 层在训练中每次更新时，将输入向量按一定比率随机地设置为 0，起到防止过拟合的作用（一般令 dropout = 0.2）；

Dropout 是解决神经网络模型过拟合的好办法。Dropout 随机地将输入的向量中的元素置为 0。将元素置为 0 相当于主动抛弃了一部分特征，强迫模型基于不完整的特征进行学习，而每次特征又不一样，所以避免模型过度学习某些特征，得到避免过拟合的效果。

SpatialDropout 是对 Dropout 的改进。传统 Dropout 随机将元素置为 0，而 SpatialDropout 每次将一整列元素置为 0，保证输入发生很大的变化，从而起到正则化的作用。

在 Keras 中可以直接调用 SpatialDropout1D 实现，参数 rate 为设置为 0 的比率，一般 rate 设置为 0.2。

- LSTM()是由存储器单元组成的 LSTM 循环层，包含 LSTM 特有的记忆单元；

LSTM 层的记忆单元个数也是一个会影响模型效果的参数。

- Dense()是跟随 LSTM 层并用于输出包含 10 个分类的预测的全连接层；
- 由于是多分类、单标签问题，所以激活函数设置为 softmax；
- 编译网络：
  - 训练神经网络的优化算法选用 Adam；

    Adam 优化算法是一种对随机梯度下降法（SGD）的扩展，SGD 保持一个单一的学习率 alpha，用于所有的权重更新，并且在训练过程中学习速率不会改变。

    Adam 每一个网络权重都保持一个学习速率，并随着学习的展开而单独地进行调整。该方法从梯度的第一次和第二次矩的预算来计算不同参数的自适应学习速率。
  - 由于是多分类、单标签问题，所以损失函数为分类交叉熵 categorical_crossentropy.

    针对不同分类问题采取的激活函数和损失函数有不同的选择，参考下图：

分类问题名称	输出层使用激活函数	对应的损失函数
二分类	sigmoid	binary_crossentrppy
多分类	softmax	categorical_crossentropy
多标签分类	sigmoid	binary_crossentropy

VI. 模型训练及超参数选择

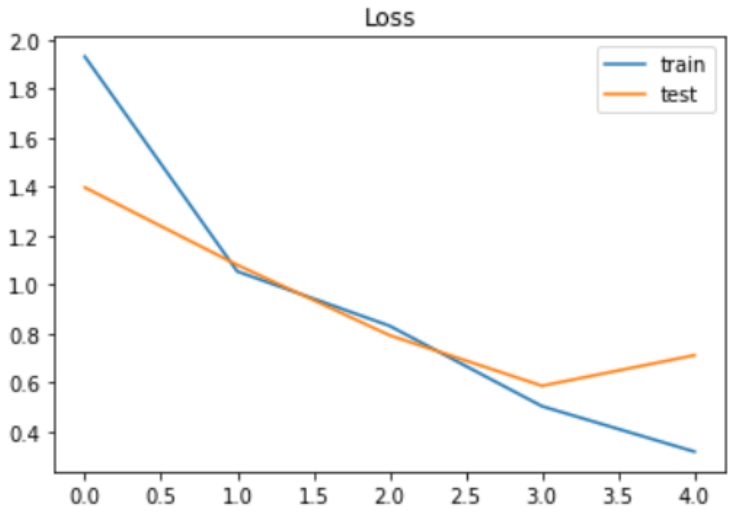
- **第 1 次超参数选择（按习惯设置）：**
  - **epochs = 5，设置 5 个训练周期**
  - **batch_size = 64**
  - **LSTM 层包含 100 个记忆单元**
  - **dropout = 0.2**
  - **激活函数使用 softmax**
  - **损失函数使用分类交叉熵 categorical_crossentropy**

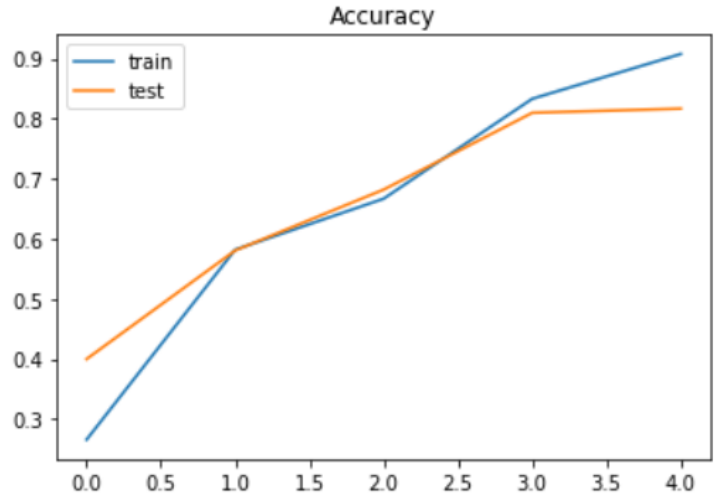
```
# 模型训练
epochs = 5
batch_size = 64

history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_split=0.1,
                    callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)])

accr = model.evaluate(X_test, Y_test)
'Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0], accr[1])
```

```
Epoch 1/5
102/102 [==============================] - 107s 1s/step - loss: 1.9303 - accuracy: 0.2659 - val_loss: 1.3956 - val_accuracy: 0.4000
Epoch 2/5
102/102 [==============================] - 97s 952ms/step - loss: 1.0527 - accuracy: 0.5819 - val_loss: 1.0792 - val_accuracy: 0.5806
Epoch 3/5
102/102 [==============================] - 94s 925ms/step - loss: 0.8302 - accuracy: 0.6668 - val_loss: 0.7916 - val_accuracy: 0.6819
Epoch 4/5
102/102 [==============================] - 95s 935ms/step - loss: 0.5018 - accuracy: 0.8330 - val_loss: 0.5853 - val_accuracy: 0.8097
Epoch 5/5
102/102 [==============================] - 96s 941ms/step - loss: 0.3163 - accuracy: 0.9076 - val_loss: 0.7102 - val_accuracy: 0.8167
25/25 [==============================] - 2s 67ms/step - loss: 0.7422 - accuracy: 0.8100

'Test set\n  Loss: 0.742\n  Accuracy: 0.810'
```

第一次训练模型 accuracy 在 0.8100，loss 在 0.7422，这样的精度是不够的。模型训练的时间在 500s 上下，还是很快的。

- **绘制损失函数趋势图和准确率趋势图：**





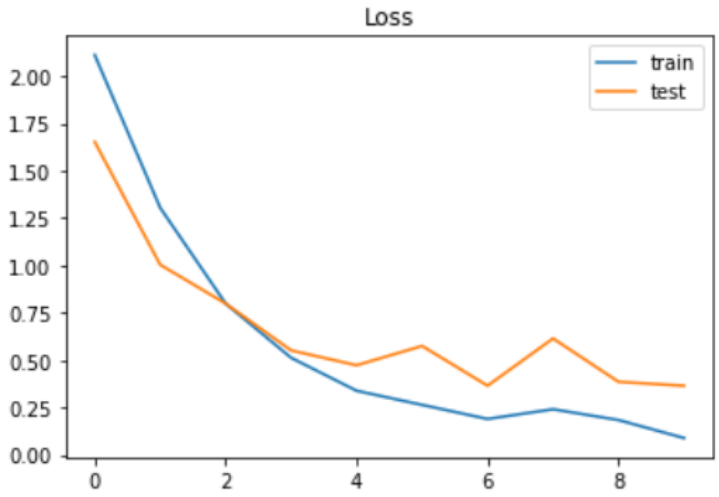
从上两幅图中我们可以看见，随着训练周期的增加，模型在训练集中损失越来越低，而在测试集中损失先从大变小最后又有些变大；模型在训练集中准确率越来越高，而在测试集中，准确率随着训练周期的增加起初越来越大但在最后准确率又有所下降。这是典型的过拟合现象。

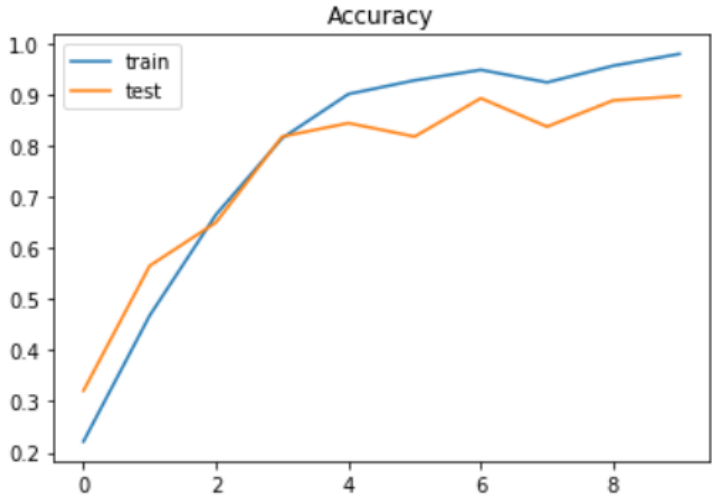
- **第 2 次超参数选择（主要扩大了 epochs 和 batch_size）：**
  - **epochs = 10，设置 10 个训练周期**
  - **batch_size = 128**
  - **LSTM 层包含 100 个记忆单元**
  - **dropout = 0.2**
  - **激活函数仍然使用 softmax**
  - **损失函数仍然使用分类交叉熵 categorical_crossentropy**

```
# 模型训练
epochs = 10
batch_size = 128

history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_split=0.1,
                    callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)])

accr = model.evaluate(X_test, Y_test)
'Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0], accr[1])
```

```
Epoch 1/10
51/51 [==============================] - 65s 1s/step - loss: 2.1124 - accuracy: 0.2215 - val_loss: 1.6534 - val_accuracy: 0.3208
Epoch 2/10
51/51 [==============================] - 61s 1s/step - loss: 1.3055 - accuracy: 0.4679 - val_loss: 1.0045 - val_accuracy: 0.5653
Epoch 3/10
51/51 [==============================] - 61s 1s/step - loss: 0.7991 - accuracy: 0.6653 - val_loss: 0.7993 - val_accuracy: 0.6500
Epoch 4/10
51/51 [==============================] - 61s 1s/step - loss: 0.5127 - accuracy: 0.8150 - val_loss: 0.5517 - val_accuracy: 0.8181
Epoch 5/10
51/51 [==============================] - 63s 1s/step - loss: 0.3387 - accuracy: 0.9012 - val_loss: 0.4733 - val_accuracy: 0.8444
Epoch 6/10
51/51 [==============================] - 62s 1s/step - loss: 0.2641 - accuracy: 0.9281 - val_loss: 0.5754 - val_accuracy: 0.8181
Epoch 7/10
51/51 [==============================] - 63s 1s/step - loss: 0.1898 - accuracy: 0.9486 - val_loss: 0.3653 - val_accuracy: 0.8931
Epoch 8/10
51/51 [==============================] - 64s 1s/step - loss: 0.2410 - accuracy: 0.9241 - val_loss: 0.6149 - val_accuracy: 0.8375
Epoch 9/10
51/51 [==============================] - 63s 1s/step - loss: 0.1841 - accuracy: 0.9566 - val_loss: 0.3858 - val_accuracy: 0.8889
Epoch 10/10
51/51 [==============================] - 63s 1s/step - loss: 0.0890 - accuracy: 0.9798 - val_loss: 0.3652 - val_accuracy: 0.8972
25/25 [==============================] - 1s 45ms/step - loss: 0.4789 - accuracy: 0.8675
```

- **绘制损失函数趋势图和准确率趋势图：**



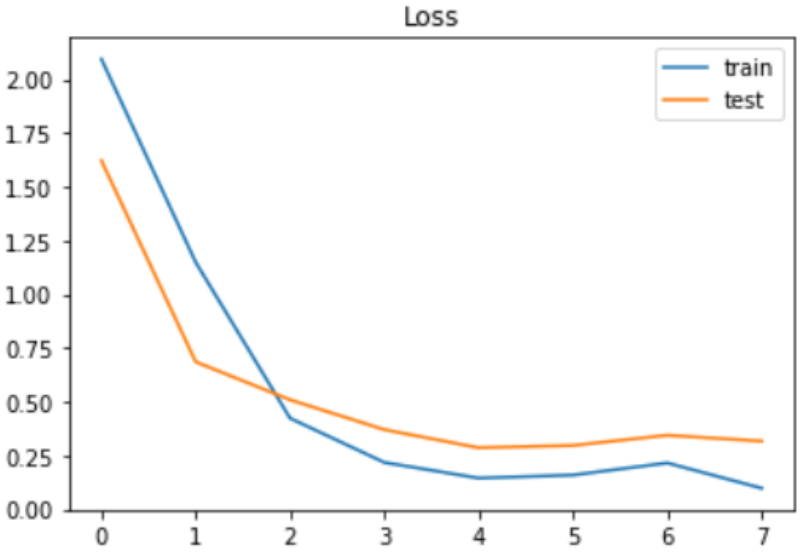


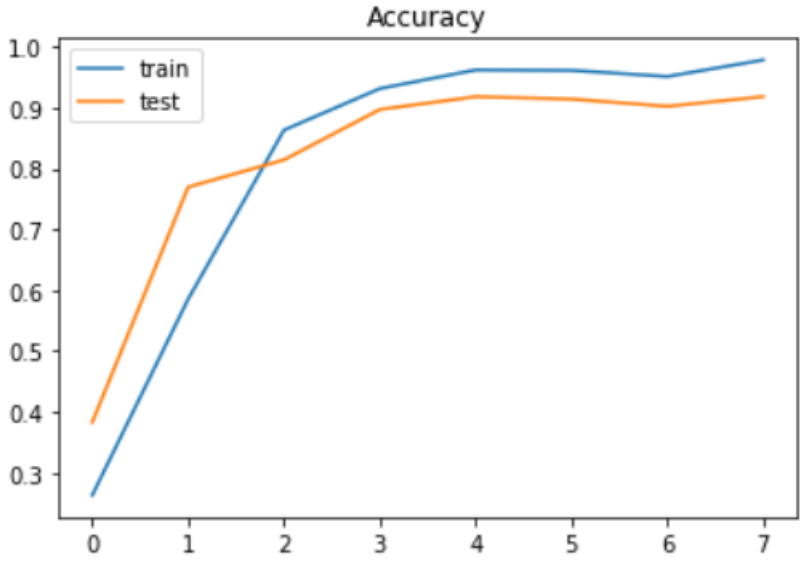
模型在训练集的表现随着训练周期的增加越来越好，然而在预测集上的表现并不平稳，过拟合问题似乎比之前严重了，但是比起第一次的模型在测试集上的 loss 和 accuracy 的表现还是更好的。模型训练时间在 630s 上下。

- **第 3 次超参数选择（在第 2 次的基础上增加了 LSTM 层的记忆单元）：**
  - **epochs = 10，设置 10 个训练周期**
  - **batch_size = 128**
  - **LSTM 层包含 200 个记忆单元**
  - **dropout = 0.2**
  - **激活函数仍然使用 softmax**
  - **损失函数仍然使用分类交叉熵 categorical_crossentropy**

```
Epoch 1/10
54/54 [==============================] - 179s 3s/step - loss: 2.0949 - accur
acy: 0.2629 - val_loss: 1.6228 - val_accuracy: 0.3829
Epoch 2/10
54/54 [==============================] - 239s 4s/step - loss: 1.1492 - accur
acy: 0.5854 - val_loss: 0.6858 - val_accuracy: 0.7697
Epoch 3/10
54/54 [==============================] - 262s 5s/step - loss: 0.4236 - accur
acy: 0.8635 - val_loss: 0.5092 - val_accuracy: 0.8145
Epoch 4/10
54/54 [==============================] - 269s 5s/step - loss: 0.2175 - accur
acy: 0.9317 - val_loss: 0.3703 - val_accuracy: 0.8974
Epoch 5/10
54/54 [==============================] - 275s 5s/step - loss: 0.1447 - accur
acy: 0.9624 - val_loss: 0.2860 - val_accuracy: 0.9184
Epoch 6/10
54/54 [==============================] - 273s 5s/step - loss: 0.1590 - accur
acy: 0.9617 - val_loss: 0.2968 - val_accuracy: 0.9145
Epoch 7/10
54/54 [==============================] - 281s 5s/step - loss: 0.2152 - accur
acy: 0.9516 - val_loss: 0.3442 - val_accuracy: 0.9026
Epoch 8/10
54/54 [==============================] - 278s 5s/step - loss: 0.0978 - accur
acy: 0.9788 - val_loss: 0.3170 - val_accuracy: 0.9184
13/13 [==============================] - 3s 247ms/step - loss: 0.3413 - accu
racy: 0.8925
```

- **绘制损失函数趋势图和准确率趋势图：**





可以看到当 LSTM 层的记忆单元从 100 个上升到 200 个时模型在训练集和预测集上的表现都有显著的提升，过拟合的问题也得到了很好的缓解。

当然付出的代价是模型训练的时间大幅上升，平均每个 epoch 在 250s 上下，一共训练模型花费 2500s 左右。

4. LSTM 模型的评估

接下来对第三次调参后的 LSTM 模型进行评估。

这里直接使用了 sklearn 中的 classification_report 函数，它是用于显示主要分类指标的文本报告，在报告中显示每个类的精确度、召回率、f1-score 等信息。

	precision	recall	f1-score	support
label 0	0.97	0.76	0.85	42
label 1	1.00	0.87	0.93	47
label 2	0.93	0.87	0.90	45
label 3	0.89	0.95	0.92	41
label 4	0.89	0.94	0.92	35
label 5	0.89	0.95	0.92	42
label 6	0.69	0.85	0.76	34
label 7	0.89	0.91	0.90	35
label 8	0.88	0.92	0.90	39
label 9	0.92	0.90	0.91	40
accuracy			0.89	400
macro avg	0.89	0.89	0.89	400
weighted avg	0.90	0.89	0.89	400

f1-score 是 P 和 R 的调和平均数，所以对模型的评估可以直接从 f1-score 上反映出来，越大越好。

从 f1-score 上看， label 1 的 f1 最大为 0.93，label 1 上得到了最好的训练，而 label 6 的 f1 最小为 0.4=76，平均 f1-score 为 0.89。

由于每个分类都拥有同样数量的语料来进行训练，因此可以排除是 label 6 的数据过少导致模型训练不够，大胆猜测是 label 6 中的文本语料的特征不明确，不便于模型计算其相似性。

5. 预测

首先读取 test.txt 中的文件：

	id	text
0	0	Considering this was shot in 1972 the video qu...
1	1	ok, I know its just random, but it seems like ...
2	2	Have used this for 5 months and have seen a di...
3	3	This app is good but I uninstallled because th...
4	4	Yes, I believe they got it right this time, to...

使用训练好的模型进行预测：

	id	text	pred
0	0	Considering this was shot in 1972 the video qu...	6
1	1	ok, I know its just random, but it seems like ...	1
2	2	Have used this for 5 months and have seen a di...	4
3	3	This app is good but I uninstallled because th...	1
4	4	Yes, I believe they got it right this time, to...	6
5	5	The end that plugs into the device, it not at ...	7
6	6	I heard the book was better but I found the do...	0
7	7	Years ago, when wrinkle development bothered m...	4
8	8	The title says it all. One of the very last M...	9
9	9	Pros: The product looks very sleek, and is ver...	7

五、 总结

对于本次实验文本多分类的问题，我使用的是**基于 Keras 的 LSTM 模型**，LSTM 是一种特殊的 RNN。

之所以采用这个模型是因为 LSTM 规避了传统 RNN 中梯度爆炸和梯度消失的问题，学习速度更快。

基于 tensorflow.keras 实现了 LSTM 模型，LSTM 的生命周期主要是 5 个步骤：定义网络、编译网络、拟合网络、评估网络、预测。

定义神经网络：神经网络在 Keras 中被定义为层序列。这些层的容器是 Sequential 类。

所以建模的第一步是创建 Sequential 类的实例。然后创建各层并按照它们应连接的顺序添加它们到模型中。其中由存储器单元组成的 LSTM 循环层称为 LSTM。通常跟随 LSTM 层并用于输出预测的全连接层被称为 Dense。神经网络的第一层必须定义预期的输入数量。输入必须是三维的，包括样本、步长和特征。各个层可以通过将 LSTM 层添加到顺序模型中来堆叠，重要的是，在叠加 LSTM 层时，我们必须为每个输入输出一个序列，而不是单个值，以便后续的 LSTM 层可以拥有所需的 3D 输入。激活函数的选择也是很重要的，在多分类单标签的分类问题中使用 softmax 函数。

编译神经网络：compile 将定义的简单层序列转换为一组高效的矩阵转换，定义模型后一定需要 compile 的步骤。compile 需要指定许多参数，是专门为了训练自己的神经网络而定制的，e.g. 用优化算法来训练网络，用损失函数来评价被优化算法最小化的网络。这里我选用了 Adam 优化算法，它是 SGD 的进阶。依然因为这是个多分类单标签的分类问题，损失函数选用的是分类交叉熵 categorical_crossentropy，从现有的一些理论知识可知这两个函数的选择是比较固定的。

拟合神经网络：拟合网络需要指定训练数据，输入矩阵 X 和匹配输出数组 y。利用反向传播算法对网络进行训练，并根据优化算法和模型编译时指定的损失函数进行优化。反向传播算法要求对网络进行特定次数的训练。每个 epoch 可以划分为一组称为批的输入-输出模式对。这定义了网络在一个 epoch 内更新权重之前所暴露的模式的数量。它也是一种效率优化，确保一次不会有太多的输入模式加载到内存中。一旦拟合，将返回一个 history 对象，该对象提供训练期间模型性能的摘要。这既包括损失，也包括在编译模型时指定的任何额外指标，记录每个 epoch。

评估神经网络：一旦网络被训练，它就可以被评估。可以根据训练数据对网络进行评估，但这不能作为预测模型提供网络性能的有用指示，因为以前已经看到了所有这些数据。在一个单独的数据集上评估网络的性能，在测试期间是看不到的，这将提供网络在预测未来不可见数据方面的性能评估。模型评估跨所有测试模式的损失，以及在编译模型时指定的任何其他度量，如分类精度。返回一个评估指标列表。

预测：当对模型的性能感到满意，就可以使用它对新数据进行预测。

遇到的问题：LSTM 作为一种 RNN 模型，一般是用于做序列预测、生成（例如投资市场时间序列预测等问题），将其用作分类比较少见，相较于 CNN 等一些神经网络感觉其本身比较难训练，参数和 hidden layer 和 units 等相关，尽管它避免了梯度爆炸和梯度消失的问题，但是 loss 降低地较慢，添加 dropout 后得到了比较好的解决，添加 weight normalization 应该也可以达到类似效果。

LSTM 特有的记忆单元也是它的核心，其中前向计算的“三个门”：遗忘门、输入门、输出门比较难以理解，好在基于 keras 可以直接生成 lstm 层并设定记忆单元的个数。