阿里云比赛-第四周-TextCNN完善

笔记本:	日常		
创建时间:	2019/10/30 19:54	**更新时间:**	2019/11/3 22:12
作者:	296645429@qq.com		
URL:	https://www.cnblogs.com/yanqiang/p/11738407.html		

1.算测试样本比例。

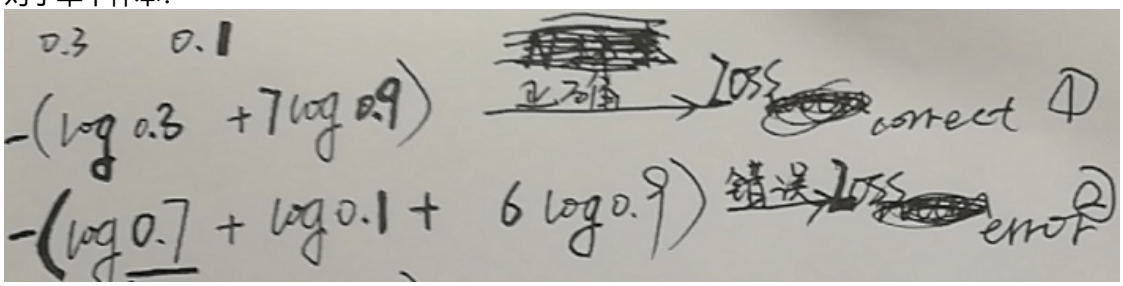
已知计算logloss的方法是：

$$\log loss = -\frac{1}{N}\sum_{i}^{N}\sum_{i}^{M}\left[y_{ij}\log(P_{ij}) + \left(1-y_{ij}\right)\log\left(1-P_{ij}\right)\right]$$

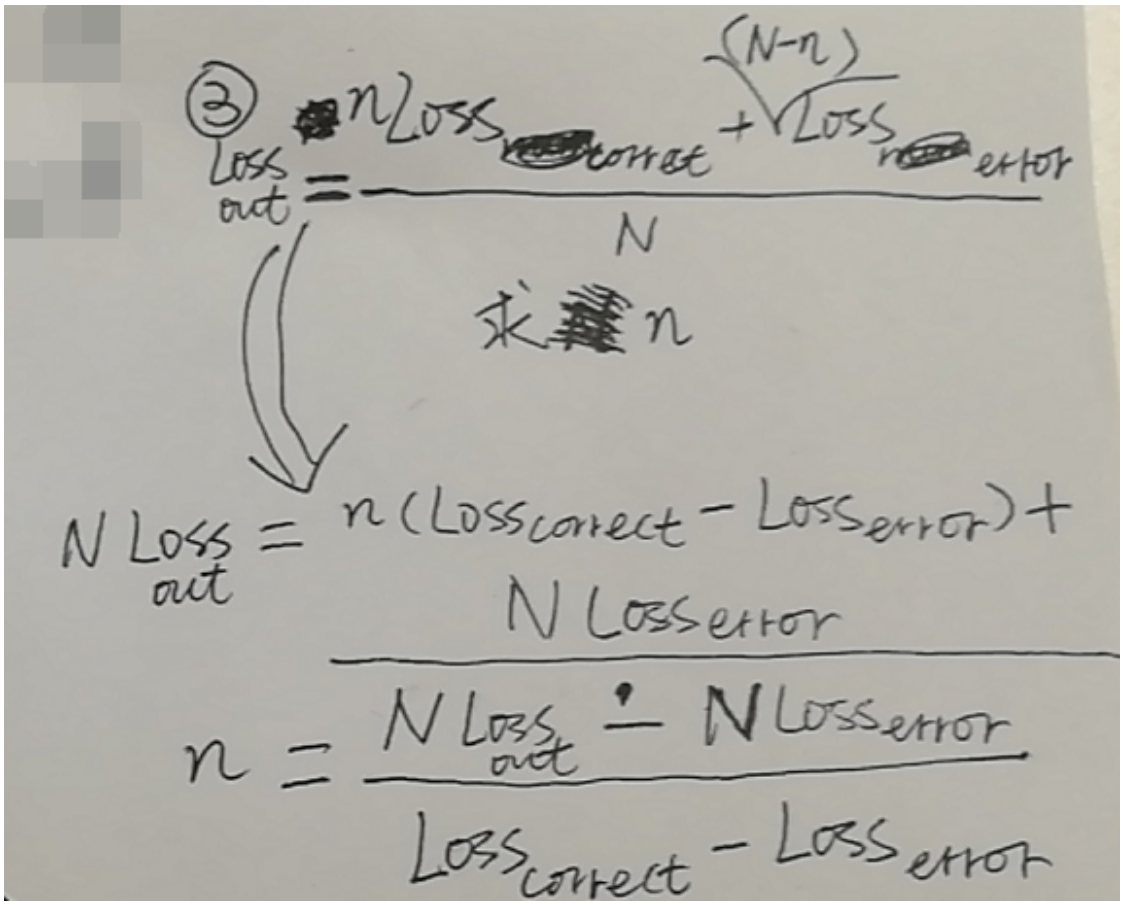
M代表分类数，N代表测试集样本数，yij代表第i个样本是否为类别j(是~1，否~0)，Pij代表选手提交的第i个样本被预测为类别j的概率(prob)，最终公布的logloss保留小数点后6位。

将表格内容设置为：可能性最大的列赋值为0.3，其他赋值为0.1【依据保持平滑过渡的思想】,8列概率综合为1符合基本条件。

对于单个样本：



对于单个列，有n个样本属于它，则这n个样本的prob赋值为0.3，其余(N-n)个样本的prob为0.1，已知对应loss值，则带入logloss化简求n。



这样依次即能求得属于各列种类的样本数。
可知测试集各类别样本数：0: 4978 1: 409 2: 643 3: 670 4: 122 5: 4288 6: 629 7: 1216
已知训练集各类别样本数：0: 4978 1: 502 2: 1196 3: 820 4: 100 5: 4289 6: 515 7: 1487
计算比例用于损失函数的惩罚：
0:1.00,1:0.81,2:0.54,3:0.82,4:1.22,5:1.00,6:1.22,7:0.82

2.keras 中模型训练class_weight

keras 中fit(self, x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None, validation_split=0.0,

validation_data=None, shuffle=True, class_weight=None, sample_weight=None, initial_epoch=0,

steps_per_epoch=None, validation_steps=None)
class_weight：字典，将不同的类别映射为不同的权值，该参数用来在训练过程中调整损失函数（只能用于训练）。该参数在处理非平衡的训练数据（某些类的训练样本数很少）时，可以使得损失函数对样本数不足的数据更加关注。
class_weight---主要针对的上数据不均衡问题，比如：异常检测的二项分类问题，异常数据仅占1%，正常数据占99%; 此时就要设置不同类对loss的影响。
sample_weight：权值的numpy array，用于在训练时调整损失函数（仅用于训练）。可以传递一个1D的与样本等长的向量用于对样本进行1对1的加权，或者在面对时序数据时，传递一个的形式为（samples，sequence_length）的矩阵来为每个时间步上的样本赋不同的权。这种情况下请确定在编译模型时添加了sample_weight_mode='temporal'。
如果仅仅是类不平衡，则使用class_weight，sample_weights则是类内样本之间还不平衡的时候使用。
例如，假设您有500个0级样本和1500个1级样本，比您在class_weight = {0：3，1：1}中输入的样本多。这给了0级三倍于1级的重量。

3.权重正则化

keras提供了三种正则化方法：

- L1：绝对值权重之和
- **L2：平方权重之和**
- L1L2:两者累加之和

```
tf.keras.regularizers.l1(l=0.01)
tf.keras.regularizers.l2(l=0.01)
tf.keras.regularizers.l1_l2(l1=0.01,l2=0.01)
```

（1）全连接层使用权重正则化

```
# 权重正则化，bias正则化（应用较少）tf.keras.layers.Dense(512,activation=tf.nn.relu,kernel_regularizer=tf.keras.regularizers.l2(l=0.001),bias_regularizer=tf.keras.regularizers.l2(l=0.001))
```

（2）卷积层使用权重正则化

```
tf.keras.layers.Conv2D(32,3,activation=tf.nn.relu,kernel_regularizer=tf.keras.regularizers.l2(l=0.0005),bias_regularizer=tf.keras.regularizers.l2(l=0.0005))
```

（3）RNN网络中使用权重正则化

```
tf.keras.layers.LSTM(32,activation=tf.nn.tanh,recurrent_regularizer=tf.keras.regularizers.l2(l=0.000001),kernel_regularizer=tf.keras.regularizers.l2(l=0.000001),bias_regularizer=tf.keras.regularizers.l2(l=0.000001))
```

（4）权重正则化使用经验

1. 最常见的权重正则化是L2正则化，数值通常是0-0.1之间，如：0.1，0.001，0.0001。
2. 找到最优的系数并不容易，需要尝试不同的权重系数，找到模型表现最平稳优秀的系数
3. L2正则化在CNN网络中，建议系数设置小一些，如：0.0005
4. 少量的权重正则对模型很重要，可以减少模型训练误差
5. LSTM网络中L2权重系数通常更小，如：10^-6

4.TPU：

https://cloud.google.com/tpu/docs/troubleshooting?hl=zh-cn#errors_in_the_middle_of_training
使用较大的批量大小，理想的**批量大小为 1024。**

每个 Cloud TPU 都由 8 个 TPU 核心组成，每个核心都有 **8GB** 的 RAM（或 HBM，即高带宽内存）。这些内存用于存储权重（可变）张量，以及梯度计算所需的中间结果张量。
如何减少内存使用量：
（1）某些优化器要求为每个权重提供额外的内存来存储更新统计信息。值得注意的是，AdamOptimizer 和 AdadeltaOptimizer 都要求为每个权重额外提供 8 个字节。AdagradOptimizer 和 MomentumOptimizer 都要求为每个权重额外提供 4 个字节。标准的 GradientDescentOptimizer 不需要任何额外的存储空间，但在最终模型准确率方面可能不如其他优化器。在训练 Transformer 模型时，实验性 **AdafactorOptimizer** 几乎不需要额外的内存，并且与**基准 Adam 优化器**的性能一样好。
如果大多数权重都是字词嵌入，则 **WordPiece 之类的技术**已被证实可以显著减少词量，同时提高执行多种任务的准确率。
即将发布的 TensorFlow 版本将提供对 16 位浮点权重和梯度的实验性支持，这会将内存需求减少一半。
（2）总批量大小应为 **64 的倍数（每个 TPU 核心 8 个），并且特征维度应为 128 的倍数。**
或
总批量大小应为 1024 的倍数（每个 TPU 核心 128 个），并且特征维度应为 8 的倍数。
使用 1024 作为批量大小以及 128 的倍数作为特征维度可以获得最佳效率，但这并非适用于所有模型。为清楚起见，“特征维度”是指全连接层的隐藏大小或卷积中的输出通道的数量。并非所有层都符合此规则，尤其是网络的第一层和最后一层。这没什么问题，并且预计大多数模型都需要一定量的填充。

```
# print(train_labels.shape)
# 5折交叉验证，将训练集切分成训练和验证集
skf = StratifiedKFold(n_splits=5)
for i, (train_index, valid_index) in enumerate(skf.split(train_apis, n_train_labels)):
    # print(i)
    # model = SequenceModel()
    # model = textcnn()
    # 设置使得样本量和batchsize（8的倍数）匹配，适应TPU
    len_train_index = len(train_index)-(len(train_index)-int(len(train_index)/batchsize)*batchsize)
    # train_index.astype(int)
    len_valid_index = len(valid_index)-(len(valid_index)-int(len(valid_index)/batchsize)*batchsize)
    # valid_index.astype(int)
    strategy = tf.contrib.tpu.TPUDistributionStrategy(
        tf.contrib.cluster_resolver.TPUClusterResolver(tpu='grpc://' + os.environ['COLAB_TPU_ADDR']))
    tpu_model = textcnn()
    tpu_model = tf.contrib.tpu.keras_to_tpu_model(tpu_model, strategy)
    # metrics默认只有loss，加accuracy后在model.evaluate(...)的返回值即有accuracy结果
    # model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    tpu_model.compile(
        optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']
    )
    #模型保存规则
    model_save_path = './my_tpumodel/my_tpumodel_{}.h5'.format(str(i))
```

（3）如何确定您的模型是否受到影响：
按照 Cloud TPU 工具：输入流水线分析器中的说明在 TensorBoard 中查看流水线分析。
如何缓解：
使用 Dataset API 加载数据时，有几种可能的缓解措施：
将您的数据以一组 tf.train.Example 结构的形式存储在 TFRecord 文件中，并使用 TFRecordDataset 加载它们。如需查看示例，请参阅 Dataset API 教程或 ResNet 教程。
使用 dataset.cache() 和/或 dataset.prefetch() 来缓冲输入数据。这可以防止因文件访问中偶尔出现的速度缓慢问题而导致产生瓶颈。
指定 dataset.map() 函数的 num_parallel_calls 参数以启用多线程 map() 操作。
离线执行数据预处理，而不是在每次训练的每个周期执行预处理。此方法的优点是一劳永逸，缺点是离线执行数据预处理的成本高昂。

5.categorical_crossentropy VS. sparse_categorical_crossentropy

如果你的 targets 是 one-hot 编码，用 categorical_crossentropy

- one-hot 编码：[0, 0, 1], [1, 0, 0], [0, 1, 0]

如果你的 tagets 是 数字编码 ，用 sparse_categorical_crossentropy

- 数字编码：2, 0, 1

6.Robust Neural Malware Detection Models for Emulation Sequence Learning

Target the core of the malicious operation by learning the presence and pattern of co-occurrence of malicious event actions

Model: LSTM+ CNNs | can handle extremely long sequences、 Convoluted Partitioning of Long Sequences approach

LSTM:可保存并记忆单元信息

CNN:smaller kernels

maxpooling:reduce dimensionality and extract significant features

7.实验

logloss	valid_acc	validloss	备注	文件名
	0.8127	0.6598	停机制设得到20代就结束	
	0.8123	0.6871		
0.837264	0.8252	0.6868	查看训练过程，loss最低能达到0.55，早停当前限制25	my_result7.csv
	0.8302	0.6959		
	0.8297	0.7395		
	0.8231	0.707		
	0.8245	0.7853		
0.838583		0.6175	api序列数据集改为1000，早停当前限制10	my_result8.csv
		0.5991		
		0.5991		
		0.6468		
		0.6551		
0.869397			api序列数据集改为7000	my_result9.csv
0.927115			api序列数据集改为7000	my_tpuresult1.csv
0.93122			api序列数据集改为7000，加入权重到loss中	my_tpuresult2.csv
0.91935		0.9-1.0	加入正则化等	my_tpuresult3.csv

目前：

网络适应tpu：batchsize=8*128（网络结构复杂不上去），xavier、加入测试集训练集比例权重、label独热平滑处理、加入正则损失。

主要代码：

```
import pickle
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Embedding, Activation, Input, Lambda,
Reshape, LSTM, RNN, CuDNNLSTM, \
    SimpleRNNCell, SpatialDropout1D, Add, Maximum
from tensorflow.keras.layers import Conv1D, Flatten, Dropout, MaxPool1D,
GlobalAveragePooling1D, concatenate, AveragePooling1D
from tensorflow.keras import optimizers
from tensorflow.keras import regularizers
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.callbacks import TensorBoard, EarlyStopping,
ModelCheckpoint
from tensorflow.keras.utils import to_categorical
import time
import numpy as np
from tensorflow.keras import backend as K
from sklearn.model_selection import StratifiedKFold
import pickle
from sklearn.model_selection import train_test_split
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
import time
import csv
import xgboost as xgb
import numpy as np
from sklearn.model_selection import StratifiedKFold
import pandas as pd
from tensorflow.keras.regularizers import l2


batchsize = 8*4
Epoch = 100
my_security_train = './my_security_train.pkl'
my_security_test = './my_security_test.pkl'
my_result = './my_tpuresult.pkl'
my_result_csv = './my_tpuresult.csv'
inputLen = 20000
# config = K.tf.ConfigProto()
# # 程序按需申请内存
# config.gpu_options.allow_growth = True
# session = K.tf.Session(config = config)


# 读取文件到变量中
with open(my_security_train, 'rb') as f:
    train_labels = pickle.load(f)
    train_apis = pickle.load(f)
with open(my_security_test, 'rb') as f:
    test_files = pickle.load(f)
    test_apis = pickle.load(f)


# print(time.strftime("%Y-%m-%d-%H-%M-%S", time.localtime()))
# tensorboard = TensorBoard('./Logs/', write_images=1, histogram_freq=1)
# print(train_labels)
# 将标签转换为空格相隔的一维数组
# train_labels = np.asarray(train_labels)
# 将标签转为独热编码
train_labels = np.eye(8)[train_labels]
# print(train_labels)
# input()
# 将独热编码平滑化
for i in range(len(train_labels)):
  for j in range(len(train_labels[i])):
    if train_labels[i][j] == 1.0:
      train_labels[i][j] = 0.995
    else:
      train_labels[i][j] = 0.005/7
# print(train_labels)
# 汇总成索引表示仅仅是为了方便切数据做交叉验证
n_train_labels = [np.argmax(i) for i in train_labels]
# print(n_train_labels)


tokenizer = Tokenizer(num_words=None,
                      filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n',
                      lower=True,
                      split=" ",
                      char_level=False)
# print(train_apis)
# 通过训练和测试数据集丰富取词器的字典，方便后续操作
tokenizer.fit_on_texts(train_apis)
# print(train_apis)
# print(test_apis)
tokenizer.fit_on_texts(test_apis)
# print(test_apis)
# print(tokenizer.word_index)
# #获取目前提取词的字典信息
# # vocal = tokenizer.word_index
train_apis = tokenizer.texts_to_sequences(train_apis)
```

```
# 通过字典信息将字符转换为对应的数字
test_apis = tokenizer.texts_to_sequences(test_apis)
# print(test_apis)
# 序列化原数组为没有逗号的数组，默认在前面填充,默认截断前面的
train_apis = pad_sequences(train_apis, inputLen, padding='post',
truncating='post')
# print(test_apis)
test_apis = pad_sequences(test_apis, inputLen, padding='post', truncating='post')




# print(test_apis.dtype)
# print(type(test_apis))


def SequenceModel():
    # Sequential()是序列模型，其实是堆叠模型，可以在它上面堆砌网络形成一个复杂的网络结构
    model = Sequential()
    model.add(Dense(32, activation='relu', input_dim=6000))
    model.add(Dense(8, activation='softmax'))
    return model




def lstm():
    my_inpuy = Input(shape=(6000,), dtype='float64')
    # 在网络第一层，起降维的作用
    emb = Embedding(len(tokenizer.word_index) + 1, 5, input_length=6000)
    emb = emb(my_inpuy)
    net = Conv1D(16, 3, padding='same', kernel_initializer='glorot_uniform')(emb)
    net = BatchNormalization()(net)
    net = Activation('relu')(net)
    net = Conv1D(32, 3, padding='same', kernel_initializer='glorot_uniform')(net)
    net = BatchNormalization()(net)
    net = Activation('relu')(net)
    net = MaxPool1D(pool_size=4)(net)


    net1 = Conv1D(16, 4, padding='same', kernel_initializer='glorot_uniform')
(emb)
    net1 = BatchNormalization()(net1)
    net1 = Activation('relu')(net1)
    net1 = Conv1D(32, 4, padding='same', kernel_initializer='glorot_uniform')
(net1)
    net1 = BatchNormalization()(net1)
    net1 = Activation('relu')(net1)
    net1 = MaxPool1D(pool_size=4)(net1)


    net2 = Conv1D(16, 5, padding='same', kernel_initializer='glorot_uniform')
(emb)
    net2 = BatchNormalization()(net2)
    net2 = Activation('relu')(net2)
    net2 = Conv1D(32, 5, padding='same', kernel_initializer='glorot_uniform')
(net2)
    net2 = BatchNormalization()(net2)
    net2 = Activation('relu')(net2)
    net2 = MaxPool1D(pool_size=4)(net2)


    net = concatenate([net, net1, net2], axis=-1)
    net = CuDNNLSTM(256)(net)
    net = Dense(8, activation='softmax')(net)
    model = Model(inputs=my_inpuy, outputs=net)
    return model
```

```
def textcnn():
    kernel_size = [1, 3, 5, 7, 9, 11, 13]
    acti = 'relu'
    # 可看做一个文件的api集为一句话，然后话中的词总量是6000
    my_input = Input(shape=(inputLen,), dtype='int32')
    emb = Embedding(len(tokenizer.word_index) + 1, 128, input_length=inputLen)
(my_input)
    emb = SpatialDropout1D(0.2)(emb)


    net = []
    for kernel in kernel_size:
        # 128个卷积核
        con = Conv1D(8, kernel, activation=acti, padding="same",
kernel_regularizer=l2(0.0005))(emb)
        # 默认输出最后一维是通道数
        con = MaxPool1D(2)(con)
        net.append(con)
    # print(net)
    # input()
    net = concatenate(net, axis=-1)
    # net = concatenate(net)
    # print(net)
    # input()
    net = Flatten()(net)
    net = Dropout(0.5)(net)
    net = Dense(256, activation='relu', kernel_regularizer=l2(l=0.001))(net)
    net = Dropout(0.5)(net)
    net = Dense(8, activation='softmax', kernel_regularizer=l2(l=0.001))(net)
    model = Model(inputs=my_input, outputs=net)
    return model


test_result = np.zeros(shape=(len(test_apis),8))


# print(train_apis.shape)
# print(train_labels.shape)
# 5折交叉验证，将训练集切分成训练和验证集
skf = StratifiedKFold(n_splits=5)
for i, (train_index, valid_index) in enumerate(skf.split(train_apis,
n_train_labels)):
    # print(i)
    # model = SequenceModel()
    # model = textcnn()
    # 设置使得样本量和batchsize（8的倍数）匹配，适应TPU
    len_train_index = len(train_index)-(len(train_index)-
int(len(train_index)/batchsize)*batchsize)
    # train_index.astype(int)
    len_valid_index = len(valid_index)-(len(valid_index)-
int(len(valid_index)/batchsize)*batchsize)
    # valid_index.astype(int)
    strategy = tf.contrib.tpu.TPUDistributionStrategy(
        tf.contrib.cluster_resolver.TPUClusterResolver(tpu='grpc://' +
os.environ['COLAB_TPU_ADDR']))
    tpu_model = textcnn()
    tpu_model = tf.contrib.tpu.keras_to_tpu_model(tpu_model, strategy)
    # metrics默认只有loss，加accuracy后在model.evaluate(...)的返回值即有accuracy结果
    # model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    tpu_model.compile(
        optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']
    )
    #模型保存规则
    model_save_path = './my_tpumodel/my_tpumodel_{}.h5'.format(str(i))
    checkpoint = ModelCheckpoint(model_save_path, save_best_only=True,
save_weights_only=True)
    #早停规则
```

```
    earlystop = EarlyStopping(monitor='val_loss', min_delta=0, patience=25,
verbose=0, mode='min', baseline=None,
                              restore_best_weights=True)
    #训练的过程会保存模型并早停
    tpu_model.fit(train_apis[train_index[:len_train_index]],
train_labels[train_index[:len_train_index]], epochs=Epoch, batch_size=batchsize,
                  validation_data=(train_apis[valid_index[:len_valid_index]],
train_labels[valid_index[:len_valid_index]]),
                  callbacks=[checkpoint, earlystop], class_weight=
{0:1.00,1:0.81,2:0.54,3:0.82,4:1.22,5:1.00,6:1.22,7:0.82})
    tpu_model.load_weights(model_save_path)
    # print(train_index, valid_index)


    cpu_model = tpu_model.sync_to_cpu()
    test_tmpapis = cpu_model.predict(test_apis)
    test_result = test_result + test_tmpapis
    K.clear_session()


# loss, acc = model.evaluate(train_apis, train_labels)
# print(loss)
# print(acc)
# print(model.predict(train_apis))


# print(test_files)
# print(test_apis)
test_result = test_result/5.0
with open(my_result, 'wb') as f:
    pickle.dump(test_files, f)
    pickle.dump(test_result, f)


# print(len(test_files))
# print(len(test_apis))




result = []
for i in range(len(test_files)):
    # #     print(test_files[i])
    #     #之前test_apis不带逗号的格式是矩阵格式，现在tolist转为带逗号的列表格式
    #     print(test_apis[i])
    #     print(test_apis[i].tolist())
    #     result.append(test_files[i])
    #     result.append(test_apis[i])
    tmp = []
    a = test_result[i].tolist()
    tmp.append(test_files[i])
    # extend相比于append可以添加多个值
    tmp.extend(a)
    #     print(tmp)
    result.append(tmp)
# print(1)
# print(result)


with open(my_result_csv, 'w') as f:
    #     f.write([1,2,3])
    result_csv = csv.writer(f)
    result_csv.writerow(["file_id", "prob0", "prob1", "prob2", "prob3", "prob4",
"prob5", "prob6", "prob7"])
    result_csv.writerows(result)


print('dowm.')
```

下周工作：

api数据集量增大、
网络结构丰富