阿里云比赛-第二周

笔记本:	日常		
创建时间:	2019/10/17 21:04	**更新时间:**	2019/10/20 23:15
作者:	296645429@qq.com		
URL:	https://zhuanlan.zhihu.com/p/29201491		

1.词袋模型
即one-hot。
假设我们总共有N个词，然后对词进行索引编码并构造一个N维零向量，如果这个文本中的某些词出现，就在该词索引值位置标记为1，表示这个文本包含这个词。
缺点：任意两个词之间都是孤立的。

2.

Tokenizer类

fit_on_text(texts) 使用一系列文档来生成token词典，texts为list类，每个元素为一个文档。
texts_to_sequences(texts) 将多个文档转换为word下标的向量形式,shape为[len(texts)，len(text)] -- (文档数，每条文档的长度)
word_index 一个dict，保存所有word对应的编号id，从1开始
示例：

```
import keras.preprocessing.text as T
from keras.preprocessing.text import Tokenizer
text1='some thing to eat'
text2='some thing to drink'
texts=[text1,text2]
print T.text_to_word_sequence(text1)  #以空格区分，中文也不例外 ['some', 'thing', 'to', 'eat']
print T.one_hot(text1,10)  #[7, 9, 3, 4] --  （10表示数字化向量为10以内的数字）
print T.one_hot(text2,10)  #[7, 9, 3, 1]
tokenizer = Tokenizer(num_words=None) #num_words:None或整数,处理的最大单词数量。少于此数的单词丢掉
tokenizer.fit_on_texts(texts)
print( tokenizer.word_counts) #[('some', 2), ('thing', 2), ('to', 2), ('eat', 1), ('drink', 1)]
print( tokenizer.word_index) #{'some': 1, 'thing': 2,'to': 3 ','eat': 4, drink': 5}
print( tokenizer.word_docs) #{'some': 2, 'thing': 2, 'to': 2, 'drink': 1,  'eat': 1}
print( tokenizer.index_docs) #{1: 2, 2: 2, 3: 2, 4: 1, 5: 1}
# num_words=多少会影响下面的结果，行数=num_words
print( tokenizer.texts_to_sequences(texts)) #得到词索引[[1, 2, 3, 4], [1, 2, 3, 5]]
print( tokenizer.texts_to_matrix(texts))  # 矩阵化=one_hot
[[ 0.,  1.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.],
[ 0.,  1.,  1.,  1.,  0.,  1.,  0.,  0.,  0.,  0.]]
```

3.

sequence模块

pad_sequences(sequences, maxlen, padding=’ pre’ , truncating=’ pre’ , value=0.) 将序列填充到maxlen长度。

```
import keras.preprocessing.sequence as S
S.pad_sequences([[1,2,3]],10,padding='post')
# [[1, 2, 3, 0, 0, 0, 0, 0, 0, 0]]
```

4.

TF-IDF

TF-IDF 用以评估一字词对于一个文档集或一个语料库中的其中一份文档的重要程度，是一种计算特征权重的方法。核心思想即，字词的重要性随着它在文档中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。有效地规避了那些高频却包含很少信息量的词。我们这里也是用TF-IDF 对文本变量进行特征提取。

作业：
训练集数据分析

```
import pandas as pd

train = pd.read_csv(r'/mnt/md0/wz/AliEvilProgramDetect/ML_Malware_detect/security_train/security_train.csv')
train.head()
```

	file_id	label	api	tid	index
0	1	5	LdrLoadDll	2488	0
1	1	5	LdrGetProcedureAddress	2488	1
2	1	5	LdrGetProcedureAddress	2488	2
3	1	5	LdrGetProcedureAddress	2488	3
4	1	5	LdrGetProcedureAddress	2488	4

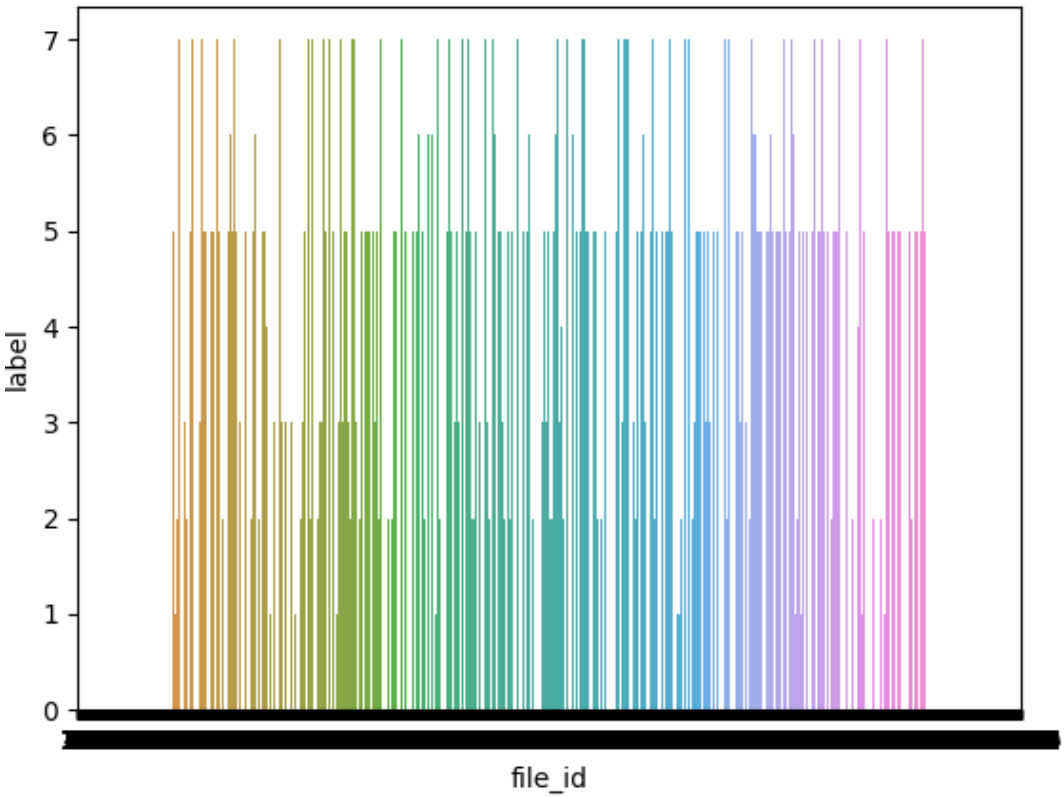
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 89806693 entries, 0 to 89806692
Data columns (total 5 columns):
file_id    int64
label      int64
api        object
tid        int64
index      int64
dtypes: int64(4), object(1)
memory usage: 3.3+ GB
```

train['file_id'].value_counts()
10811 888204
11303 791687

```
5885 537495
13468 506888
9540 443957
13016 406399
9610 391349
3545 365888
13793 329338
9147 328642
7688 293097
143 285885
8226 277294
7635 261001
11561 257231
8730 255212
1934 247519
6892 234271
5903 222801
4639 222417
9206 215564
13327 210816
12607 203810
389 189886
1345 189125
7003 182595
13733 178528
13883 178221
11634 171100
8433 160204
...
13353 2
9175 2
2578 2
547 2
12614 2
9887 2
9272 2
1941 2
7614 2
10767 2
6602 2
7489 2
7620 2
13344 2
6040 2
2561 2
10492 2
9621 1
716 1
9368 1
10610 1
12060 1
10096 1
10635 1
8877 1
12736 1
1253 1
7871 1
6643 1
6060 1
Name: file_id, Length: 13887, dtype: int64
```

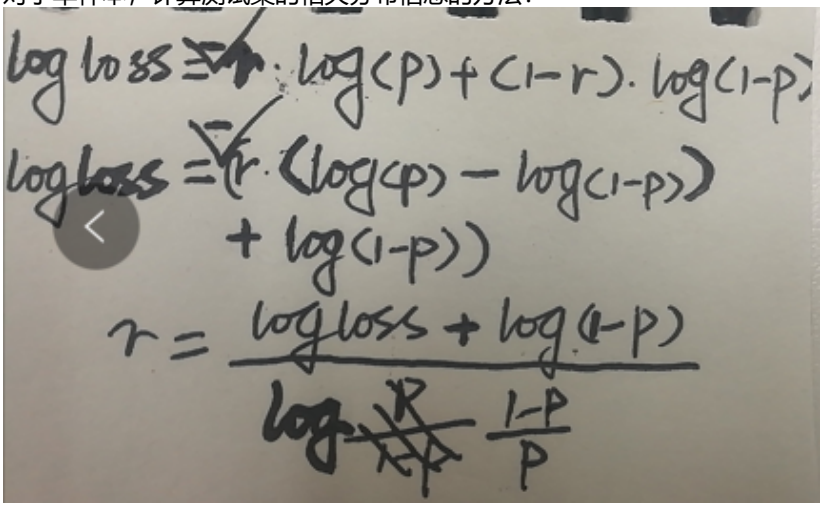
```
sns.barplot('file_id', 'label', data=train)
pit.show()
```



一般比赛使用的评价方式是**log loss**, 即逻辑回归中的损失函数. 对于这种特定的评价方式, 能用下面的方法, 探测出提交的测试集中, 正样本的比例:
得到测试集中正样本的比例之后, **一个比较有效的提高leaderboard排名的方式是**: 判断训练集和测试集中的正样本比例相差是否过大. 假设它们来自于同一个分布采样得到的, 那么就应该调整**训练集的样本比例**或者**使用改造的损失函数**来解决训练集和测试集分布不一致的问题.
本题中分数采用logloss计算公式如下:

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

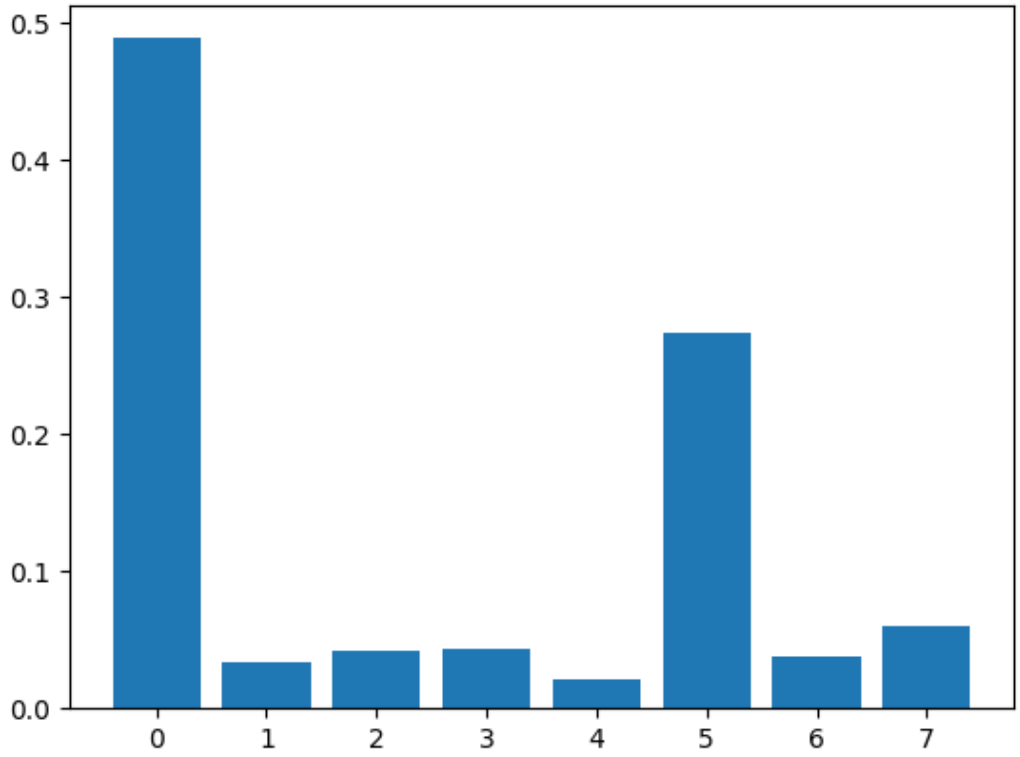
M代表分类数，N代表测试集样本数，yij代表第i个样本是否为类别j(是~1，否~0)，Pij代表选手提交的第i个样本被预测为类别j的概率(prob)，最终公布的logloss保留小数点后6位。
对于单样本，计算测试集的相关分布信息的方法:



因此，接下来思路是求取训练集各类样本比例。本题目的分类结果有8类，用了逻辑回归损失函数，所以我是先预测了一个结果，提交至官网得出log loss值为0.647898，然后我在预测结果中分别求出8类在12955个测试样本下预测值的平均值，再使用上式计算出8类的概率值r，计算它们占总概率值的份额，得到在测试集中各类数据集的份额，如下图（注意：计算时log底数选择默认值即可，所以概率值不关注，关注他们占用的份额即可）。

各列总值	5004.351783	415.8032159	718.6368394	723.6922942	85.91429709	4188.256726	550.5913034	1267.753547
各列平均值	0.386287285	0.032095964	0.055471775	0.055862007	0.006631748	0.323292684	0.042500294	0.097858244
概率值	-1.992135591	-0.136990363	-0.173236124	-0.173837115	-0.087972717	-1.116231027	-0.15328421	-0.241758109
概率值份额	0.488814219	0.033613594	0.042507288	0.042654754	0.021586038	0.273891798	0.037611647	0.059320662

numlist=
[0.488814219,0.033613594,0.042507288,0.042654754,0.021586038, 0.273891798,0.037611647,0.059320662]
plt.bar(range(len(numlist)),numlist)
plt.show()
得到测试集的label占比。



获取训练集的label比例：
去重方法1：
# 删除表中的某一行或者某一列更明智的方法是使用drop，它不改变原有的df中的数据，而是返回另一个dataframe来存放删除后的数据。
# drop函数默认删除行，列需要加axis = 1
train_low = train.drop(['api', 'tid', 'index'], axis=1)
# 一行元素全部相同时才去除
train_low = train_low.drop_duplicates()

Out[4]:

	file_id	label
0	1	5
6786	2	2
7602	3	0
8065	4	0
10111	5	0
20113	6	0
25114	7	0
30396	8	0
30514	9	0

去重方法2：

```
# 根据'file_id', 'label'组合列删除重复项，默认保留第一个出现的值组合。
train_drop = train.drop_duplicates(['file_id', 'label'])
train_drop['label'].value_counts()
```

```
0    4978
5    4289
7    1487
2    1196
3     820
6     515
1     502
4     100
Name: label, dtype: int64
```

```
import seaborn as sns
sns.countplot(x='label', data = train_drop)
import matplotlib.pyplot as plt
plt.show()
```

得到训练集的label占比