

Kaggle Tabular Playground Series(June)

竞赛报告

邢存远, 191300064@smail.nju.edu.cn

2021 年 7 月 1 日

摘要

我们记录在参加 2021 年 Kaggle 第六次月赛时的比赛成绩, 问题求解和建模的思路以及在比赛过程中的模型探究。

1 比赛规则简述

Kaggle Tabular Playground Series 是由 Kaggle 提供的练手竞赛, 每月举办 1 次, 时长一个月, 难度相对较低. 该竞赛的目标是基于输入特征预测数据类别, 是一个多分类任务。我们参加的是今年六月的比赛, 地址: <https://www.kaggle.com/c/tabular-playground-series-jun-2021>。

比赛的每支队伍不超过 3 人, 每支队伍每天只能提交五次。Kaggle 会对提交的预测结果进行评分, 在比赛结束之前, 会有一个 public leaderboard 对参赛队伍的最好成绩进行排名, 而在比赛结束后, Kaggle 会用另外的测试数据来测试我们的模型, 然后进行排名, 排名榜称为 private leaderboard。这样, 即使我们在 public leaderboard 获得了很好的名次, 但如果你的模型过拟合, 缺少泛化能力, 那么到了 private leaderboard, 排名则有可能后退。因此我们需要做好交叉验证。

2 数据初探与性能评估

2.1 数据集的基本情况

从官网上我们可以下载比赛需要的带标签测试集和无标签测试集的 csv 文件: train.csv 和 test.csv。通过使用 Python 的 pandas 模块, 我们可以了解数据的基本情况:

测试集, 200000 条数据, 75 个特征, 9 个类别:

id	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	...	feature_66	feature_67	feature_68	feature_69	feature_70	feature_71	feature_72	feature_73	feature_74	target	
0	0	0	6	1	0	0	0	0	7	0	...	0	0	0	0	0	0	2	0	0	Class_6	
1	0	0	0	0	0	0	0	0	0	0	...	2	0	0	0	0	0	0	1	0	Class_6	
2	0	0	0	0	0	1	0	3	0	0	...	0	0	0	0	1	0	0	0	0	Class_2	
3	0	0	7	0	1	5	2	2	0	1	...	0	4	0	2	2	0	4	3	0	Class_8	
4	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Class_2	
...
199995	0	1	6	0	1	32	0	6	0	0	...	0	1	1	0	0	0	4	1	0	Class_6	
199996	0	2	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Class_6	
199997	1	2	0	0	0	2	0	1	8	4	...	1	0	1	1	1	0	1	0	0	Class_8	
199998	0	0	2	0	2	1	0	0	3	1	...	0	0	3	2	1	0	0	1	0	Class_7	
199999	5	4	0	10	0	1	0	0	12	2	...	0	0	2	1	0	0	2	3	1	Class_8	

200000 rows x 75 columns

训练集：100000 条数据，75 个特征：

id	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	...	feature_65	feature_66	feature_67	feature_68	feature_69	feature_70	feature_71	feature_72	feature_73	feature_74	
200000	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
200001	1	2	0	0	0	0	0	0	0	0	...	3	1	3	0	0	0	0	0	3	0	0
200002	0	1	7	1	0	0	0	0	0	0	...	3	0	0	0	0	3	0	2	0	0	0
200003	0	0	0	4	3	1	0	0	0	0	...	0	0	0	1	0	0	0	4	0	0	0
200004	0	0	0	5	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1	0
...
299995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
299996	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0
299997	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0	0	0	0	0	0	0	2
299998	0	0	0	0	2	2	0	0	1	0	...	1	0	0	0	0	1	0	1	0	0	0
299999	0	0	0	0	0	0	6	0	1	11	...	0	2	0	0	0	0	0	0	0	0	0

100000 rows x 75 columns

接下来是对训练集数据进行统计学方面的观察。我们先用 pandas 模块的 describe 函数查看训练集数据的情况：

	count	mean	std	min	25%	50%	75%
feature_0	200000.0	0.972710	3.941836	0.0	0.0	0.0	1.0
feature_1	200000.0	1.168365	3.993407	0.0	0.0	0.0	1.0
feature_2	200000.0	2.219325	6.476570	0.0	0.0	0.0	1.0
feature_3	200000.0	2.296735	7.551858	0.0	0.0	0.0	1.0
feature_4	200000.0	0.793530	2.935785	0.0	0.0	0.0	0.0
...
feature_70	200000.0	1.219210	4.826003	0.0	0.0	0.0	1.0
feature_71	200000.0	0.806895	2.458741	0.0	0.0	0.0	1.0
feature_72	200000.0	1.282925	4.261420	0.0	0.0	0.0	1.0
feature_73	200000.0	2.940210	10.784650	0.0	0.0	0.0	1.0
feature_74	200000.0	0.632005	3.925310	0.0	0.0	0.0	0.0

我们会发现数据集的一个特点是，很多数据项都是 0：不少特征超过一半的数值都是 0，这是否说明，数据的差别可能只是 0 与非 0 的差别，而不是数值大小？

特征多（75 个）也是该数据集的一个显著特点，随之而来的是学习时间的大幅度延长。

2.2 特征工程

对于比赛而言，数据处理、特征工程与模型选择、模型训练同样重要。而且在大多数情况下，针对某些特征明显的问题，所有参赛者使用的模型可能都是类似的，最终成绩在很大程度上取决于特征工程。

我们现在已经遇到了两个问题：

- 数据特征数量多；
- 数据分布畸形：0 的数量太多。

对于第一个问题，我们可以采用随机选择特征：每次随机选择特征集合的子集进行训练，然后将结果集成；而对于第二个问题，笔者的想法是将数据基于阈值为 1 的二值化。有趣的是，在浏览 Kaggle 交流区的时候发现了和我有相同想法的选手。

2.3 性能评估

我们的任务是通过学习测试数据，然后对测试集进行预测，采用对数损失对学习效果进行评估：

$$\text{loss} = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^M y_{ij} p_{ij}$$

我们对上式进行解释：我们对测试集进行分类时，是依据概率进行分类：

$$P = [p_{ij}]_{N \times M}$$

此问题中 $N = 100000$, $M = 9$, p_{ij} 表示数据集中第 i 条数据 \mathbf{x}_i 是第 j 类的概率，我们上传给 Kaggle 评估的就是矩阵 P 的 csv 形式；而 y_{ij} 是一个指示器：对于数据 i ，如果分类是正确的，也就是正确答案是 j 类，而你提交的预测结果恰为 j ，也就是预测正确，那么 $y_{ij} = 1$ ，否则为 0；而 y_{ij} 是一个指示器：对于数据 i ，如果分类是正确的，也就是正确答案是 j 类，而你提交的预测结果恰为 j ，也就是预测正确，那么 $y_{ij} = 1$ ，否则为 0：

$$\text{Given } \mathbf{x}_i \text{ belongs to class } c : y_{ij} = \begin{cases} 1 & \text{if } c = j \\ 0 & \text{else} \end{cases}$$

3 模型选择与评估

3.1 第一次提交

就下来就是选择模型 → 学习 → 测试 → 提交的机械过程，我们先选择一个简单的模型进行试提交，同时也将这一流程的代码框架确定下来。我们选用 sklearn 的集成学习模块中的随机森林模型进行学习。

随机森林是一种集成学习方法，由多个决策树构成，用投票法决定输出概率或结果：假设我们有 n 个决策树，对于一个 C 分类问题， $c_i (i = 1, 2, \dots, n)$ 表示第 i 棵决策树的对数据 \mathbf{x} 的分类结果，从而对于整个随机森林， \mathbf{x} 是第 c 类的概率：

$$\Pr(\mathbf{x} \in c) = \frac{1}{n} \sum_{i=1}^n \mathbb{I}(c_i = c)$$

其中 $\mathbb{I}(p)$ 是一个指示器函数：如果 p 命题为真则函数值为 1，否则为 0。

在 sklearn 中，对应的分类器模型是 sklearn.ensemble.RandomForestClassifier，我们可以写出分类任务的 Python 代码：

```
from sklearn.ensemble import RandomForestClassifier
import pandas as pd

# We don't mention how to read and process data here
model = RandomForestClassifier()
model.fit(train_x, train_y)
proba = model.predict_proba(test_x.values)
```

```

output = pd.DataFrame({
    'id': test_x.index,
    'Class_1': proba[:, 0],
    'Class_2': proba[:, 1],
    'Class_3': proba[:, 2],
    'Class_4': proba[:, 3],
    'Class_5': proba[:, 4],
    'Class_6': proba[:, 5],
    'Class_7': proba[:, 6],
    'Class_8': proba[:, 7],
    'Class_9': proba[:, 8],
})
output.id += 200000 # In order to satisfy format required by kaggle
output.to_csv("my_submission.csv", index=False)

```

然后我们去官网提交观察结果 (submit predictions) :

Name	Submitted	Wait time	Execution time	Score
my_submission.csv	a few seconds ago	1 seconds	1 seconds	2.01416
Complete				

我们在 public leaderboard 上的排名是 854/921, 而第一名的成绩是 1.74389 (6 月 23 日排行榜)。作为参考, 比赛方设计了一个基准: p_{ij} 都是 0.1111, 也就是 $\frac{1}{9}$, 即随机猜测, 它的分数是 2.19722, 说明我们还差的很远。

由于每天只能提交 5 次, 我们必须得自己设计评估函数, 将给定的测试数据进行训练, 测试和评估, 选定最合适的参数后训练再提交。

3.2 数据划分与性能评估的设计

我们首先得对有标签数据集划分成训练集和测试集:

```

def data_split(path="train.csv", frac=0.8):
    # 根据path获取数据

    # 随机打乱数据 (shuffle)

    # 定义bound
    bound = int(data_len * frac)

    # 划分
    df_train = df.iloc[:bound]
    df_test = df.iloc[bound:]
    return (
        df_train.iloc[:, :-1], # 训练集特征向量
        df_train['target'], # 训练集标签
        df_test.iloc[:, :-1], # 测试集特征向量
        df_test['target'], # 测试集标签
    )

```

)

在训练模型后，我们需要对已有模型进行评估，为此我们定义一个 ‘evaluation’ 函数，计算对应的 loss:

```

from numpy as np
from sklearn.preprocessing import OneHotEncoder

def evaluation(result, test_y):
    result = result.values
    test_y_array = test_y.values.reshape(-1, 1)

    result[result == 0.0] = 0.0000001

    enc = OneHotEncoder()
    enc.fit(test_y_array)
    y = enc.transform(test_y_array).toarray()

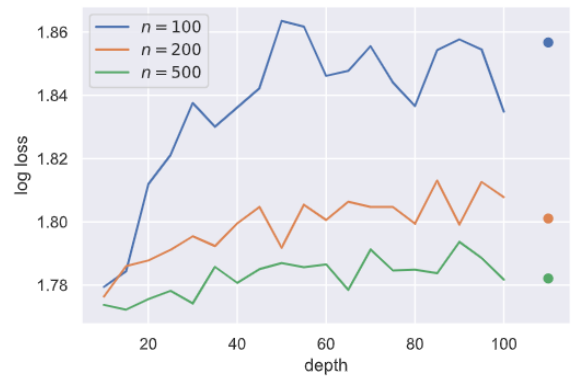
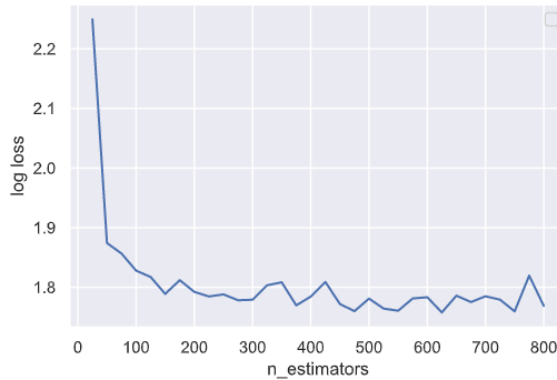
    loss = -np.sum(np.log(result) * y) / result.shape[0]
    return loss

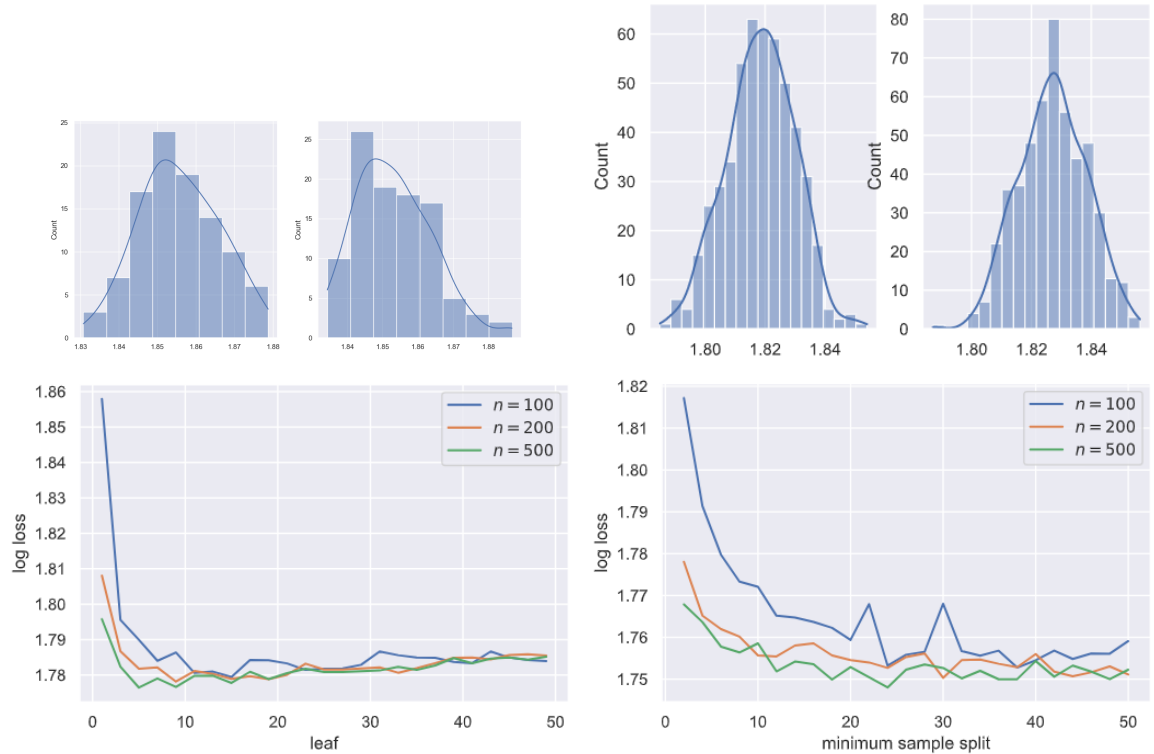
```

值得注意的是，我们将概率预测为 0 的 p_{ij} 项改成极小的 0.0000001，是为了在后面的对数运算中避免 $\log 0$ 的异常，我们相信在 kaggle 的判定程序中也有类似的处理，但惩罚可能比较大。这一函数也为后面的调参环节提供数值依据。

3.3 随机森林的参数调整

我们对 RF 的参数进行了一系列探究：





最终确定了一个较适合该学习任务的参数集合，但由于随机森林本身的不足，导致对数损失始终下不去。我们必须考虑其他模型和方法。

3.4 对结果进行集成

在阅读 Kaggle 上选手的比赛经验分享时，看到了这样的做法：分别用 xgboost 和 catboost 对数据进行学习和预测，得到了两个预测概率表 P_1 和 P_2 ，然后进行平均处理：

$$P_{\text{new}} = \frac{(P_1 + P_2)}{2}$$

然后就可以得到更好的分数。我们对这一做法进行实验上的验证：

```
list_rf, list_ada, list_gbd, list_ens = [], [], [], []

for i in range(20):
    ...
    训练20次，比较各自方法和集成后的区别
    ...

    logloss1, logloss2, logloss3, logloss4 = 0, 0, 0, 0

for j in range(5):
    train_x, train_y, test_x, test_y = data_split(frac=1 / 3)
    model1 = RandomForestClassifier()
    model2 = AdaBoostClassifier()
    model3 = GradientBoostingClassifier()
```

```
model1.fit(train_x, train_y)
model2.fit(train_x, train_y)
model3.fit(train_x, train_y)

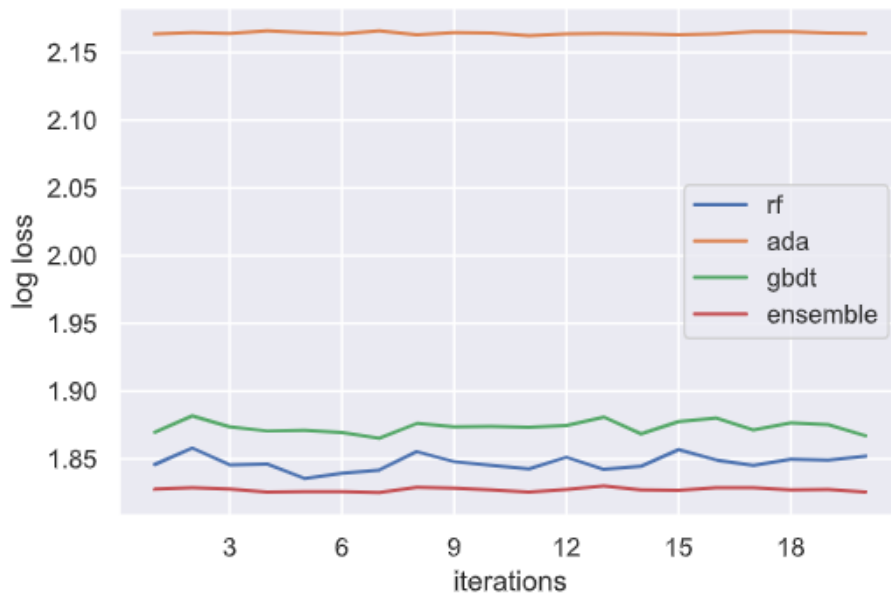
output1 = test(test_x, model1)
output2 = test(test_x, model2)
output3 = test(test_x, model3)
output4 = (output1 + output2 + output3) / 3

logloss1 += evaluation(output1, test_y)
logloss2 += evaluation(output2, test_y)
logloss3 += evaluation(output3, test_y)
logloss4 += evaluation(output4, test_y)

print("%d iter %d cross validation" % (i + 1, j + 1))

list_rf.append(logloss1 / 5)
list_ada.append(logloss2 / 5)
list_gbdt.append(logloss3 / 5)
list_ens.append(logloss4 / 5)
```

我们选用了三种分类器：Adaboost 分类器，RF 分类器和 GradientBoost 分类器进行 20 次 5 折交叉验证实验，同时将三种模型的结果进行平均化处理，并计算这四种情况对数损失，并将结果绘制出来：



我们可以发现，将三种模型的结果进行组合，并不代表 log loss 的平均，而是增强。这也启示我们在后面的模型中使用类似方法以取得更好成绩。

4 选用新的模型

Kaggle 比赛主要使用的模型有两种：集成学习和深度学习，上面提到的 RandomForest 就属于集成学习方法。我们接下来会选择其他的集成学习方法，包括常用的深度神经网络进行分类。

4.1 深度森林

深度森林是周志华老师等提出的新型深度学习模型。2021 年 2 月 1 日，深度森林软件包 DF21 开源发布，它拥有比其他基于决策树的集成学习方法更好的性能，更少的超参数，并且无需大量的调参，训练效率高。

深度森林提供了和上面 `sklearn.ensemble.RandomForestClassifier` 类似的接口和参数，方便我们使用，我们可以用和上面类似的方式探究和调整参数。但更好的方法是参考其文档：<https://deep-forest.readthedocs.io/en/stable/>，告诉我们如何通过调参增加模型的复杂度：

参数名	解释
<code>n_estimators</code>	每个级联层的森林数目
<code>n_trees</code>	每个森林的决策树数目
<code>max_layers</code>	级联层的最大数目

而在使用该框架的过程中，以及和同学之间交流时，都发现深度森林由于其本身是依赖于决策，也就是条件判断，而不是矩阵运算，所以无法使用目前的 GPU 加速。

4.2 深度神经网络

我们这里将目光转向深度学习方法，借助 Pytorch 和 TensorFlow 最为流行的深度神经框架，可以对数据集进行分类。

4.2.1 关于 Pytorch 及其使用

PyTorch 是一个以 Python 优先的深度学习框架，不仅能够实现强大的 GPU 加速，同时还支持动态神经网络，这是现在很多主流框架比如 Tensorflow 等都不支持的。PyTorch 既可以看做加入了 GPU 支持的 numpy，同时也可以看成一个拥有自动求导功能的强大的深度神经网络，除了 Facebook 之外，它还已经被 Twitter、CMU 和 Salesforce 等机构采用。

我们并不想把本节搞成一个《Pytorch 入门讲解》，因此这里只会有我们如何用它去实现比赛任务的过程与代码。

我们第一步是构建一个程序框架来进行分类任务，先从一个简单的单隐藏层神经网络开始。和前面不一样，Pytorch 强迫我们将数据集封装成一个 DataLoader 已进行后面的训练：

```
import torch
```

```
# 先将数据转换为张量
```



```
X_train_tensor = torch.from_numpy(X_train.values).type(torch.FloatTensor)
y_train_tensor = torch.from_numpy(y_train.values).type(torch.LongTensor)
X_test_tensor = torch.from_numpy(X_test.values).type(torch.FloatTensor)

# 构造train_loader
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
)
```

然后是写神经网络，一个 $75 \times 100 \times 9$ ，用 ReLU 作激活函数的神经网络：

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.input = nn.Linear(
            in_features=75,
            out_features=100,
        )
        self.hidden = nn.Linear(
            in_features=100,
            out_features=100,
        )
        self.output = nn.Linear(
            in_features=100,
            out_features=9,
        )

    def forward(self, x):
        x = self.input(x)
        x = F.relu(x)
        x = self.hidden(x)
        x = F.relu(x)
        x = self.output(x)
        return x
```

定义网络，损失函数和算法：

```
net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.0001)
```

然后进行 5 个批次的训练：

```
for epoch in range(5):
```

```
running_loss = 0.0
for batch_idx, data in enumerate(train_loader):
    # get inputs and labels
    inputs, labels = data
    # zero the parameter gradients
    optimizer.zero_grad()
    # forward + backward + optimize
    outputs = net(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    # print statistics
    running_loss += loss.item()
```

然后通过

```
F.softmax(torch_train()(X_test_tensor)).detach().numpy()
```

可以得到对应的输出及其概率，也就是我们想要的目标数据。提交，分数为 1.77509，由于模型复杂度不高，这样的结果是在预期之内的。

4.2.2 接下来的方向

我们接下来会使用神经网络的技巧，比如卷积，嵌入和 drop-out 处理等操作，继续提高成绩，同时也不能忘记前面提到的集成方法。

5 代码工程结构

我们将本次比赛使用过的算法进行一个总结，并将其放置在 Github 上（链接<https://github.com/Kaslanarian/code>）。在文件夹中，我们将数据获取与处理、算法训练、以及模型评估等步骤解耦，并将使用到的三种模型：Deep-forest、深度神经网络、带有嵌入层和 drop-out 机制的神经网络模型分别放在的 forest.py、torch_classify.py 和 tf_classify.py。将三种模型在 main.py 中一起训练，并将结果集成，最后的结果会被保存在 my_submission.csv 文件中，我们最后提交的正是该概率预测。

6 测试结果以及最终结果简述

我们的工作思路是先对不同模型各自调参，训练出较好的模型后，再调整集成模型的参数，也就是概率权重。


对于深度森林模型，我们一开始调整的是 n_estimators 和 n_trees，但很快分类器的性能便到达了 1.76 附近的瓶颈。然后是对 max_depth 和 max_layers 进行调整，然后性能确实获得了提升，在 1.755 附近徘徊，最后是对多个训练结果进行线性组合，成功将成绩提高到了 1.75 附近。

而对于神经网络，在实现了上文提到的单隐层模型之后，便打算简单通过层数的堆叠实现更好的效果。我好奇地用 20 层神经网络测试，训练后的结果是 6.02！我将其归因为欠拟合，那么我们为什么不增大训练轮次呢？事实上笔者的设备不足以支持这种耗时长规模大的运算，只好作罢，另寻出路。

在 Kaggle 社区看到了关于神经网络的思路，其中有大量的 trick，包括我们上面提到的嵌入，卷积和 drop-out 等，也让我认识到 DL 并不仅是隐藏层的堆砌。利用嵌入 + drop-out，我们得以缓解过拟合问题，并将成绩进入了 1.75 大关。

后面的工作就是对集成权重进行调整。该工作虽然枯燥，但也能引发思考，比如为什么这种操作会带来效果的增强？我的理解是该操作会矫正分类器对某一种数据的偏好，使其分类更“公平”。还有的一些想法是，除了线性组合，是否有其他的结合方法？

比赛的最终结果于七月一日公布，以 private leaderboard 的成绩为准，下面是我的成绩：

10	▲230	grayjay		1.73919	10	1d
11	▲7	JustNJUNobody		1.73920	5	10h
12	▲7	KasInarian Welt		1.73920	20	10h
13	▲91	Koearn		1.73921	57	3d
14	▲13	Yue Sun		1.73921	140	1d

令人欣喜的是我们的 private leaderboard 成绩相比于 public leaderboard，其实是提升了的。