Task 1.4 设计/实现新的特征选择算法

ID: 14 Name: 中科大飙车队

任务描述

- 自主任务
- 目标是设计/实现新的特征选择的算法

背景

原生的NNI只支持两种特征选择的方法，分别是GradientFeatureSelector和GBDTSelector. 而且不支持特征组合的方法。相比NAS模块已经支持了大量的架构搜索的方法，特征选择的方法相比之下显得较少。在Task1.3中，我们已经学习了开源项目中特征选择的方法，希望能做进一步的探索和改进。另外，特征选择消耗的算力也相对较少，比较适合我们进行改进和优化。

优化方法

总的来看，我们的优化方法借鉴了处理MAB(multi-armed bandits)问题的思想，利用了MAB问题常见的处理方法UCB算法对特征选择的算法进行了优化。我们首先简要对MAB问题进行描述，之后再对UCB算法进行一些介绍，最后介绍我们如何在本问题中应用UCB算法对特征优化算法进行建模。

MAB问题简介

MAB问题的本质是一个选择的问题。MAB的意思是多臂老虎机，多臂老虎机主要处理的场景是：在赌场中有K台老虎机，每台老虎机均有不等的概率收益或损失，赌客需要选择在老虎机上进行操作的顺序和次数，以追求达到自己收益的最大化。MAB的关键是Exploration和Exploitation的trade off。赌客既要保证自己能有足够的次数对所有的老虎机进行探索以便找出其中收益最高的一个，又要避免在探索中浪费太多的金钱，尽早找到一个收益高的老虎机获得不错的收益。

UCB算法简介

UCB算法是处理MAB问题的一种常见的算法。UCB算法中并没有引入随机性，具有较好的鲁棒性。UCB算法的处理流程如下所示

首先定义第t轮第i台老虎机的回报为，$n_{i,t}$表示在前t时刻第i台老虎机被选中的次数。

$$\mu_{i,t} = \frac{\sum_{s=1:I_s=i}^{t} r_s}{n_{i,t}}$$

接下来定义第t轮第i台老虎机的UCB值

$$UCB_{i,t} = \mu_{i,t} + \sqrt{\frac{\ln t}{n_{i,t}}}$$

我们假设有N台老虎机，在前N轮中我们依次选取每台老虎机，在接下来的所有轮次中，我们选取在上一轮中UCB值最高的老虎机。

我们的建模方法

特征选择问题与UCB算法要解决的问题既有很多相似之处，又有一些不同。UCB算法和特征选择算法都是解决在未知收益(的概率分布)的情形下做出累积最优的选择问题，然而，UCB算法只能解决对单个老虎机的收益选择，而特征选择算法是要选取出一组最优特征，而这一组最优特征往往之间还有着复杂的相互作用和关系。特征选择算法对Exploration的要求更高，因为要选择的特征多，特征之间的相互作用复杂，需要对特征空间进行更加全面的搜索。

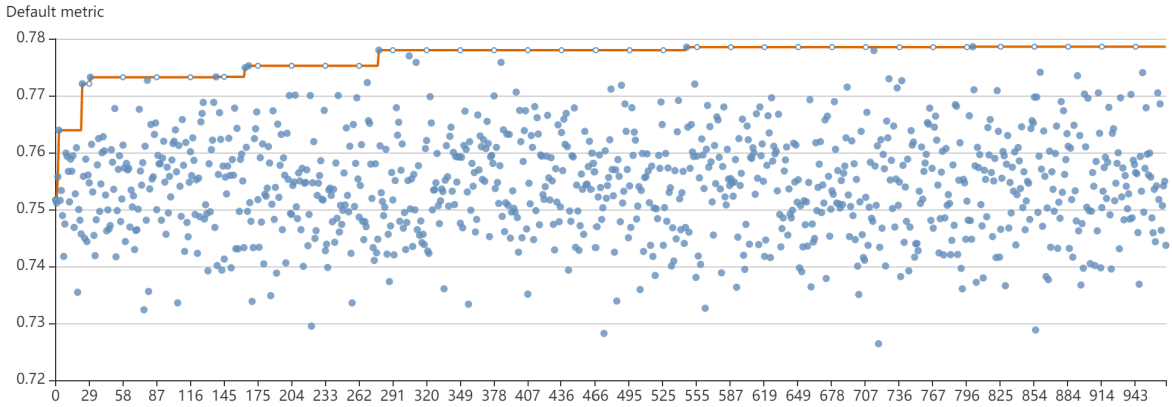
我们的算法基于Task1.3中的代码进行了实验。Task1.3中利用了特征的feature score作为抽样选取特征的依据。我们将特征的feature score看作该特征到目前为止所得到的回报，再加上UCB算法中涉及到探索次数的一项$\sqrt{\frac{\ln t}{n_{i,t}}}$。这里不同的是我们没有按照UCB算法的要求，选择UCB值最大的M个特征，因为这可能导致搜索范围变得过小。我们将UCB值做归一化之后作为采样概率。关键部分的实现如下所示

```
def generate_parameters(self, parameter_id, **kwargs):
        """Returns a set of trial graph config, as a serializable object.
        parameter_id : int
        """
        self.count += 1
        if self.count == 0:
            return {'sample_feature': []}
        else:
            if self.count == 1:
                self.sample_count =
np.zeros_like(np.array(self.estimate_sample_prob))+1.0
            sample_p = np.array(self.estimate_sample_prob) /
np.sum(self.estimate_sample_prob)
            logger.info(str(sample_p))
            sample_size = min(128, int(len(self.candidate_feature) *
self.feature_percent))
            candidate_length = list(np.arange(0,len(self.candidate_feature)))
            sample_p += np.sqrt(np.log(self.count)/self.sample_count)
            sample_p = sample_p / np.sum(sample_p)
            sample_index = np.random.choice(
                candidate_length,
                size = sample_size,
                p = sample_p,
                replace = False
                )
            sample_feature = []
            for i in sample_index:
                sample_feature.append(self.candidate_feature[i])
                self.sample_count[i] += 1
            gen_feature = list(sample_feature)
            r = {'sample_feature': gen_feature}
            return r
```

实验效果

我们迭代了1000轮，得到的结果如下所示

Trial No.	ID	Duration	Status	Default metric ↓
801	F2XAH	4s	SUCCEEDED	0.778581
545	VrnoM	11s	SUCCEEDED	0.778515
279	QuY9Y	4s	SUCCEEDED	0.777979
714	Odw9l	4s	SUCCEEDED	0.777912
305	Q6otr	6s	SUCCEEDED	0.776974
384	hxEtX	10s	SUCCEEDED	0.775836
311	PwBs2	5s	SUCCEEDED	0.775836
167	Zn7V0	5s	SUCCEEDED	0.775233
164	gO20Y	4s	SUCCEEDED	0.774898
859	BmRmt	3s	SUCCEEDED	0.774094



相比于task1.3.1中得到的结果，经过优化后的算法在准确率上略有提升。另外，在Task1.3中，后200次迭代都没有得到进入Top10的结果，而优化后的算法能在较为靠后的轮次得到不错的结果，体现出随着搜索的深入找到了更好的结果。

我们将详细的实验结果整理在下表中

Dataset	baseline auc	automl in task 1.3.2 auc	automl in task 1.4 auc	number of cat	number of num
train-tiny	0.751658	0.778515	0.778581	13	13
bank-additional	0.933209	0.944439	0.944626	10	10
bank-additional-full	0.951733	0.955124	0.954771	10	10

总结

- 使用了借鉴UCB算法思想的新的特征搜索算法后，确实可以一定程度上提升特征搜索的效果。
- Task1.3.2中我们也给出了优化特征搜索算法的方法，都可以在一定程度上提升搜索的效果。

- 特征搜索算法的效果的衡量需要考虑以下几点因素：要在尽可能短的轮次搜索到一个准确率相对不错的特征组合，以便在算力和时间有限的情况下达到较好的特征组合；随着迭代轮次的增加，靠后的轮次选择出的特征应该平均来看更好，否则无法起到优化特征选择算法的意义。
- 设计出好的特征组合的方式对于提升特征搜索算法的效果有着重要意义。