入门 | 从结构到性能，一文概述XGBoost、Light GBM和CatBoost的同与不同

机器之心 2018-03-18 13:16

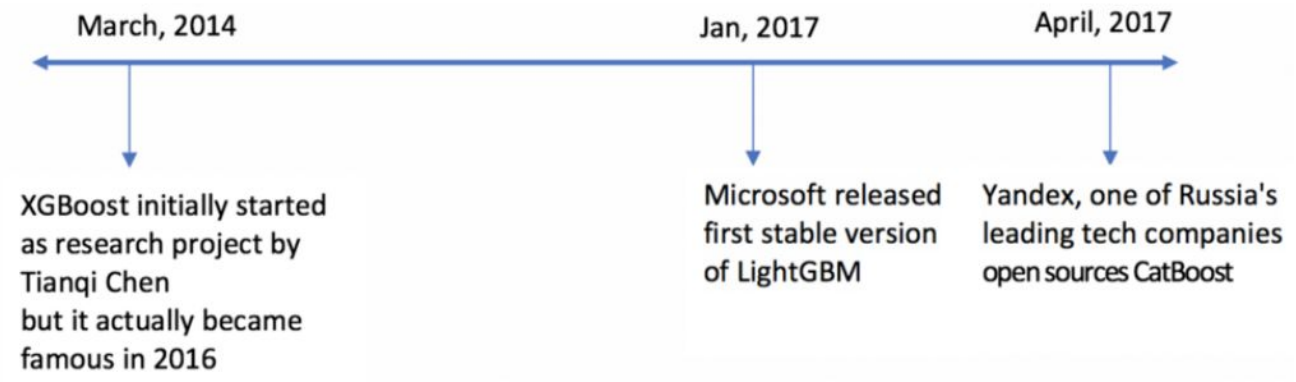
选自Medium

机器之心编译

参与：刘天赐、黄小天

尽管近年来神经网络复兴并大为流行，但是 boosting 算法在训练样本量有限、所需训练时间较短、缺乏调参知识等场景依然有其不可或缺的优势。本文从算法结构差异、每个算法的分类变量时的处理、算法在数据集上的实现等多个方面对 3 种代表性的 boosting 算法 CatBoost、Light GBM 和 XGBoost 进行了对比；虽然本文结论依据于特定的数据集，但通常情况下，XGBoost 都比另外两个算法慢。

最近，我参加了 kaggle 竞赛 WIDS Datathon，并通过使用多种 boosting 算法，最终排名前十。从那时开始，我就对这些算法的内在工作原理非常好奇，包括调参及其优劣势，所以有了这篇文章。尽管最近几年神经网络复兴，并变得流行起来，但我还是更加关注 boosting 算法，因为在训练样本量有限、所需训练时间较短、缺乏调参知识的场景中，它们依然拥有绝对优势。



- 2014 年 3 月，XGBOOST 最早作为研究项目，由陈天奇提出
- 2017 年 1 月，微软发布首个稳定版 LightGBM
- 2017 年 4 月，俄罗斯顶尖技术公司 Yandex 开源 CatBoost

由于 XGBoost（通常被称为 GBM 杀手）已经在机器学习领域出现了很久，如今有非常多详细论述它的文章，所以本文将重点讨论 CatBoost 和 LGBM，在下文我们将谈到：

- 算法结构差异
- 每个算法的分类变量时的处理
- 如何理解参数
- 算法在数据集上的实现
- 每个算法的表现

LightGBM 和 XGBoost 的结构差异

在过滤数据样例寻找分割值时，LightGBM 使用的是全新的技术：基于梯度的单边采样（GOSS）；而 XGBoost 则通过预分类算法和直方图算法来确定最优分割。这里的样例（instance）表示观测值/样本。

首先让我们理解预分类算法如何工作：

- 对于每个节点，遍历所有特征
- 对于每个特征，根据特征值分类样例
- 进行线性扫描，根据当前特征的基本信息增益，确定最优分割

- 选取所有特征分割结果中最好的一个

简单说，直方图算法在某个特征上将所有数据点划分到离散区域，并通过使用这些离散区域来确定直方图的分割值。虽然在计算速度上，和需要在预分类特征值上遍历所有可能的分割点的预分类算法相比，直方图算法的效率更高，但和 GOSS 算法相比，其速度仍然更慢。

为什么 GOSS 方法如此高效?

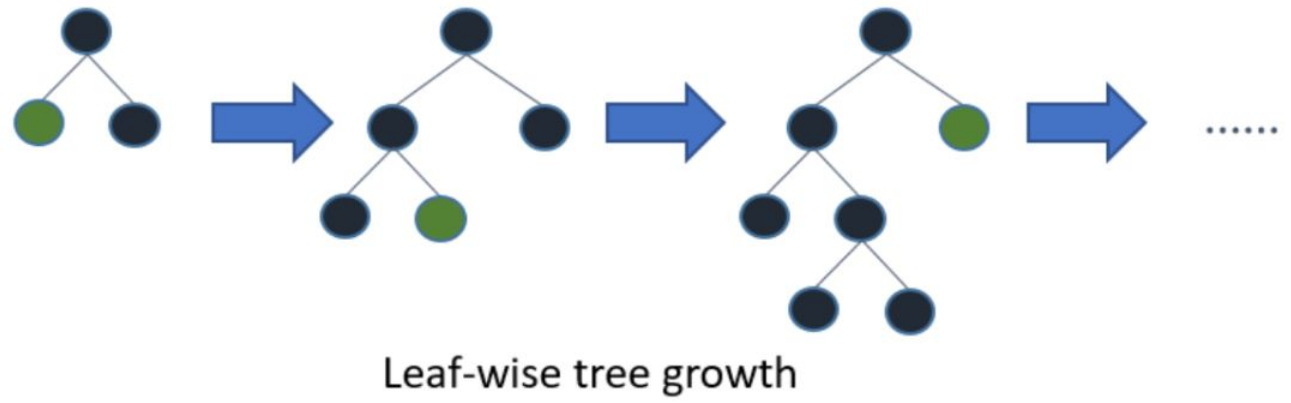
在 Adaboost 中，样本权重是展示样本重要性的很好的指标。但在梯度提升决策树（GBDT）中，并没有天然的样本权重，因此 Adaboost 所使用的采样方法在这里就不能直接使用了，这时我们就需要基于梯度的采样方法。

梯度表征损失函数切线的倾斜程度，所以自然推理到，如果在某些意义上数据点的梯度非常大，那么这些样本对于求解最优分割点而言就非常重要，因为算其损失更高。

GOSS 保留所有的大梯度样例，并在小梯度样例上采取随机抽样。比如，假如有 50 万行数据，其中 1 万行数据的梯度较大，那么我的算法就会选择（这 1 万行梯度很大的数据+x% 从剩余 49 万行中随机抽取的结果）。如果 x 取 10%，那么最后选取的结果就是通过确定分割值得到的，从 50 万行中抽取的 5.9 万行。

在这里有一个基本假设：如果训练集中的训练样例梯度很小，那么算法在这个训练集上的训练误差就会很小，因为训练已经完成了。

为了使用相同的数据分布，在计算信息增益时，GOSS 在小梯度数据样例上引入一个常数因子。因此，GOSS 在减少数据样例数量与保持已学习决策树的准确度之间取得了很好的平衡。



高梯度/误差的叶子，用于 LGBM 中的进一步增长

每个模型是如何处理属性分类变量的?

CatBoost

CatBoost 可赋予分类变量指标，进而通过独热最大量得到独热编码形式的结果（独热最大量：在所有特征上，对小于等于某个给定参数值的不同的数使用独热编码）。

如果在 CatBoost 语句中没有设置「跳过」，CatBoost 就会将所有列当作数值变量处理。

注意，如果某一列数据中包含字符串值，CatBoost 算法就会抛出错误。另外，带有默认值的 int 型变量也会默认被当成数值数据处理。在 CatBoost 中，必须对变量进行声明，才可以让算法将其作为分类变量处理。

```
from catboost import CatBoostRegressor
# Initialize data
cat_features = [0,1,2]
train_data = [["a","b",1,4,5,6],["a","b",4,5,6,7],["c","d",30,40,50,60]]
test_data = [["a","b",2,4,6,8],["a","d",1,4,50,60]]
train_labels = [10,20,30]
# Initialize CatBoostRegressor
model = CatBoostRegressor(iterations=2, learning_rate=1, depth=2)
# Fit model
model.fit(train_data, train_labels, cat_features)
```

对于可取值的数量比独热最大量还要大的分类变量，CatBoost 使用了一个非常有效的编码方法，这种方法和均值编码类似，但可以降低过拟合情况。它的具体实现方法如下：

1. 将输入样本集随机排序，并生成多组随机排列的情况。
2. 将浮点型或属性值标记转化为整数。
3. 将所有的分类特征值结果都根据以下公式，转化为数值结果。

$$avg\_target = \frac{countInClass + prior}{totalCount + 1}$$

其中 CountInClass 表示在当前分类特征值中，有多少样本的标记值是「1」；Prior 是分子的初始值，根据初始参数确定。TotalCount 是在所有样本中（包含当前样本），和当前样本具有相同的分类特征值的样本数量。

可以用下面的数学公式表示：

$$\text{Let } \sigma = (\sigma_1, \ldots, \sigma_n) \text{ be the permutation, then } x_{\sigma_p,k} \text{ is substituted with}$$
$$\frac{\sum_{j=1}^{p-1} [x_{\sigma_j,k} = x_{\sigma_p,k}] Y_{\sigma_j} + a \cdot P}{\sum_{j=1}^{p-1} [x_{\sigma_j,k} = x_{\sigma_p,k}] + a}, \quad (1)$$

LightGBM

和 CatBoost 类似，LighGBM 也可以通过使用特征名称的输入来处理属性数据；它没有对数据进行独热编码，因此速度比独热编码快得多。LGBM 使用了一个特殊的算法来确定属性特征的分割值。

Specific feature names and categorical features:

```
train_data = lgb.Dataset(data, label=label, feature_name=['c1', 'c2', 'c3'],
↪categorical_feature=['c3'])
```

注意，在建立适用于 LGBM 的数据集之前，需要将分类变量转化为整型变量；此算法不允许将字符串数据传给分类变量参数。

XGBoost

和 CatBoost 以及 LGBM 算法不同，XGBoost 本身无法处理分类变量，而是像随机森林一样，只接受数值数据。因此在将分类数据传入 XGBoost 之前，必须通过各种编码方式：例如标记编码、均值编码或独热编码对数据进行处理。

超参数中的相似性

所有的这些模型都需要调节大量参数，但我们只谈论其中重要的。以下是将不同算法中的重要参数按照功能进行整理的表格。

Function	XGBoost	CatBoost	Light GBM
Important parameters which control overfitting	1. **learning_rate or eta** – optimal values lie between 0.01-0.2 2. **max_depth** 3. **min_child_weight:** similar to min_child leaf; default is 1	1. **Learning_rate** 2. **Depth** - value can be any integer up to 16. Recommended - [1 to 10] 3. No such feature like min_child_weight 4. **l2-leaf-reg**: L2 regularization coefficient. Used for leaf value calculation (any positive integer allowed)	1. **learning_rate** 2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM 3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	1. **cat_features**: It denotes the index of categorical features 2. **one_hot_max_size**: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255)	1. **categorical_feature:** specify the categorical features we want to use for training our model
Parameters for controlling speed	1. **colsample_bytree**: subsample ratio of columns 2. **subsample**: subsample ratio of the training instance 3. **n_estimators**: maximum number of decision trees; high value can lead to overfitting	1. **rsm:** Random subspace method. The percentage of features to use at each split selection 2. No such parameter to subset data 3. **iterations**: maximum number of trees that can be built; high value can lead to overfitting	1. **feature_fraction:** fraction of features to be taken for each iteration 2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. **num_iterations:** number of boosting iterations to be performed; default=100

实现

在这里，我使用了 2015 年航班延误的 Kaggle 数据集，其中同时包含分类变量和数值变量。这个数据集中一共有约 500 万条记录，因此很适合用来同时评估比较三种 boosting 算法的训练速度和准确度。我使用了 10% 的数据：50 万行记录。

以下是建模使用的特征：

- 月、日、星期：整型数据
- 航线或航班号：整型数据
- 出发、到达机场：数值数据
- 出发时间：浮点数据
- 到达延误情况：这个特征作为预测目标，并转为二值变量：航班是否延误超过 10 分钟
- 距离和飞行时间：浮点数据

```
import pandas as pd, numpy as np, time
from sklearn.model_selection import train_test_split

data = pd.read_csv("flights.csv")
data = data.sample(frac = 0.1, random_state=10)
```

```
data = data[["MONTH","DAY","DAY_OF_WEEK","AIRLINE","FLIGHT_NUMBER","DESTINATION_AIRPORT",
                "ORIGIN_AIRPORT","AIR_TIME", "DEPARTURE_TIME","DISTANCE","ARRIVAL_DELAY"]]
data.dropna(inplace=True)

data["ARRIVAL_DELAY"] = (data["ARRIVAL_DELAY"]>10)*1

cols = ["AIRLINE","FLIGHT_NUMBER","DESTINATION_AIRPORT","ORIGIN_AIRPORT"]
for item in cols:
    data[item] = data[item].astype("category").cat.codes +1

train, test, y_train, y_test = train_test_split(data.drop(["ARRIVAL_DELAY"], axis=1), data["ARRIVAL_DELAY"],
                                                random_state=10, test_size=0.25)
```

XGBoost

```
import xgboost as xgb
from sklearn import metrics

def auc(m, train, test):
    return (metrics.roc_auc_score(y_train,m.predict_proba(train)[:,1]),
                            metrics.roc_auc_score(y_test,m.predict_proba(test)[:,1]))

# Parameter Tuning
model = xgb.XGBClassifier()
param_dist = {"max_depth": [10,30,50],
              "min_child_weight" : [1,3,6],
              "n_estimators": [200],
              "learning_rate": [0.05, 0.1,0.16],}
grid_search = GridSearchCV(model, param_grid=param_dist, cv = 3,
                                    verbose=10, n_jobs=-1)
grid_search.fit(train, y_train)

grid_search.best_estimator_

model = xgb.XGBClassifier(max_depth=50, min_child_weight=1,  n_estimators=200,\
                          n_jobs=-1 , verbose=1,learning_rate=0.16)
model.fit(train,y_train)

auc(model, train, test)
```

Light GBM

```
import lightgbm as lgb
from sklearn import metrics

def auc2(m, train, test):
    return (metrics.roc_auc_score(y_train,m.predict(train)),
                            metrics.roc_auc_score(y_test,m.predict(test)))

lg = lgb.LGBMClassifier(silent=False)
param_dist = {"max_depth": [25,50, 75],
              "learning_rate" : [0.01,0.05,0.1],
              "num_leaves": [300,900,1200],
              "n_estimators": [200]
             }
grid_search = GridSearchCV(lg, n_jobs=-1, param_grid=param_dist, cv = 3, scoring="roc_auc", verbose=5)
grid_search.fit(train,y_train)
grid_search.best_estimator_

d_train = lgb.Dataset(train, label=y_train)
params = {"max_depth": 50, "learning_rate" : 0.1, "num_leaves": 900,  "n_estimators": 300}

# Without Categorical Features
model2 = lgb.train(params, d_train)
auc2(model2, train, test)

#With Catgeorical Features
cate_features_name = ["MONTH","DAY","DAY_OF_WEEK","AIRLINE","DESTINATION_AIRPORT",
                "ORIGIN_AIRPORT"]
model2 = lgb.train(params, d_train, categorical_feature = cate_features_name)
auc2(model2, train, test)
```

CatBoost

在对 CatBoost 调参时，很难对分类特征赋予指标。因此，我同时给出了不传递分类特征时的调参结果，并评估了两个模型：一个包含分类特征，另一个不包含。我单独调整了独热最大量，因为它并不会影响其他参数。

```
import catboost as cb
cat_features_index = [0,1,2,3,4,5,6]

def auc(m, train, test):
    return (metrics.roc_auc_score(y_train,m.predict_proba(train)[:,1]),
                          metrics.roc_auc_score(y_test,m.predict_proba(test)[:,1]))

params = {'depth': [4, 7, 10],
          'learning_rate' : [0.03, 0.1, 0.15],
         'l2_leaf_reg': [1,4,9],
         'iterations': [300]}
cb = cb.CatBoostClassifier()
cb_model = GridSearchCV(cb, params, scoring="roc_auc", cv = 3)
cb_model.fit(train, y_train)

With Categorical features
clf = cb.CatBoostClassifier(eval_metric="AUC", depth=10, iterations= 500, l2_leaf_reg= 9, learning_rate= 0.15)
clf.fit(train,y_train)
auc(clf, train, test)

With Categorical features
clf = cb.CatBoostClassifier(eval_metric="AUC",one_hot_max_size=31, \
                          depth=10, iterations= 500, l2_leaf_reg= 9, learning_rate= 0.15)
clf.fit(train,y_train, cat_features= cat_features_index)
auc(clf, train, test)
```

结语

	XGBoost	Light BGM		CatBoost	
Parameters Used	max_depth: 50 learning_rate: 0.16 min_child_weight: 1 n_estimators: 200	max_depth: 50 learning_rate: 0.1 num_leaves: 900 n_estimators: 300		depth: 10 learning_rate: 0.15 l2_leaf_reg= 9 iterations: 500 one_hot_max_size = 50	
Training AUC Score	0.999	Without passing indices of categorical features	Passing indices of categorical features	Without passing indices of categorical features	Passing indices of categorical features
		0.992	0.999	0.842	0.887
Test AUC Score	0.789	0.785	0.772	0.752	0.816
Training Time	970 secs	153 secs	326 secs	180 secs	390 secs
Prediction Time	184 secs	40 secs	156 secs	2 secs	14 secs
Parameter Tuning Time (for 81 fits, 200 iteration)	500 minutes	200 minutes		120 minutes	

为了评估模型，我们应该同时考虑模型的速度和准确度表现。

请记住，CatBoost 在测试集上表现得最好，测试集的准确度最高（0.816）、过拟合程度最小（在训练集和测试集上的准确度很接近）以及最小的预测和调试时间。但这个表现仅仅在有分类特征，而且调节了独热最大量时才会出现。如果不利用 CatBoost 算法在这些特征上的优势，它的表现效果就会变成最差的：仅有 0.752 的准确度。因此我们认为，只有在数据中包含分类变量，同时我们适当地调节了这些变量时，CatBoost 才会表现很好。

第二个使用的是 XGBoost，它的表现也相当不错。即使不考虑数据集包含有转换成数值变量之后能使用的分类变量，它的准确率也和 CatBoost 非常接近了。但是，XGBoost 唯一的问题是：它太慢了。尤其是对它进行调参，非常令人崩溃（我用了 6 个小时来运行 GridSearchCV——太糟糕了）。更好的选择是分别调参，而不是使用 GridSearchCV。

最后一个模型是 LightGBM，这里需要注意的一点是，在使用 CatBoost 特征时，LightGBM 在训练速度和准确度上的表现都非常差。我认为这是因为它在分类数据中使用了一些修正的均值编码方法，进而导致了过拟合（训练集准确率非常高：0.999，尤其是和测试集准确率相比之下）。但如果我们像使用

XGBoost 一样正常使用 LightGBM，它会比 XGBoost 更快地获得相似的准确度，如果不是更高的话（LGBM—0.785, XGBoost—0.789）。

最后必须指出，这些结论在这个特定的数据集下成立，在其他数据集中，它们可能正确，也可能并不正确。但在大多数情况下，XGBoost 都比另外两个算法慢。

所以，你更喜欢哪个算法呢？

原文地址：https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

本文为机器之心编译，转载请联系本公众号获得授权。

✂----------