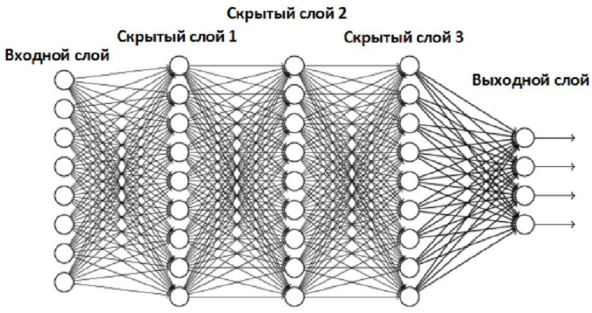
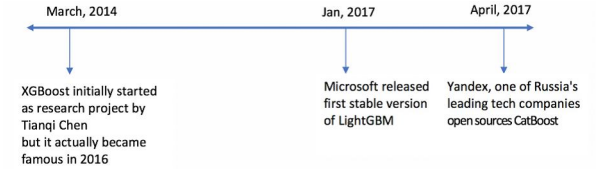


CatBoost против Light GBM против XGBoost

Дата публикации Mar 13, 2018

Кто победит в этой войне предсказаний и какой ценой? Давайте исследуем.

Недавно я участвовал в этом конкурсе Kaggle (WIDS Datathon от Stanford), где мне удалось попасть в топ-10 с использованием различных алгоритмов повышения. С тех пор мне было очень любопытно, как работает каждая модель, включая настройку параметров, плюсы и минусы, и поэтому решил написать этот блог. Несмотря на недавнее повторное появление и популярность нейронных сетей, я сосредоточен на улучшении алгоритмов, потому что они все еще более полезны в режиме ограниченных данных обучения, небольшого времени обучения и небольшого опыта для настройки параметров.



Поскольку XGBoost (часто называемый GBM Killer) уже долгое время находится в мире машинного обучения со множеством статей, посвященных этому, эта статья будет больше сфокусирована на CatBoost & LGBM. Ниже приведены темы, которые мы рассмотрим

- Структурные различия
- Обработка категориальных переменных по каждому алгоритму
- Понимание параметров
- Реализация на наборе данных
- Производительность каждого алгоритма

Структурные различия в LightGBM и XGBoost

LightGBM использует новую технику односторонней выборки на основе градиента (GOSS) для фильтрации экземпляров данных для нахождения значения разделения, в то время как XGBoost использует предварительно отсортированный алгоритм и алгоритм на основе гистограммы для вычисления наилучшего разделения. Здесь случаи означают наблюдения / образцы.

Во-первых, давайте разберемся, как работает предварительная сортировка

- Для каждого узла перечислите все функции
- Для каждой функции сортируйте экземпляры по значению
- Используйте линейное сканирование, чтобы выбрать лучшее разделение по этой характеристикеполучение информации
- Возьмите лучшее сплит-решение по всем функциям

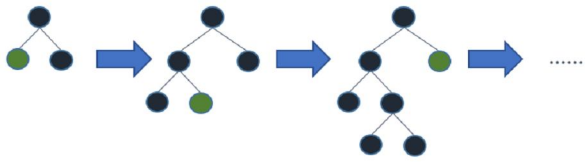
Проще говоря, алгоритм на основе гистограммы разделяет все точки данных для объекта на дискретные элементы и использует эти элементы для поиска значения разделения гистограммы. Несмотря на то, что он эффективнее, чем предварительно отсортированный алгоритм в скорости обучения, который перечисляет все возможные точки разделения на предварительно отсортированные значения признаков, он все еще отстает от GOSS в плане скорости.

Так что же делает этот метод GOSS эффективным?
В AdaBoost вес образца служит хорошим индикатором важности образцов. Однако в дереве решений с градиентным ускорением (GBDT) нет собственных весов выборки, и поэтому методы выборки, предложенные для AdaBoost, не могут быть применены напрямую. Здесь идет выборка на основе градиента.

> Градиент представляет наклон тангенса функции потерь, поэтому логически, если в некотором смысле градиент точек данных велик, эти точки важны для нахождения оптимальной точки разделения, поскольку они имеют более высокую ошибку

GOSS сохраняет все экземпляры с большими градиентами и выполняет случайную выборку для экземпляров с маленькими градиентами. Например, допустим, у меня есть 500 000 строк данных, где 10 000 строк имеют более высокий градиент. Поэтому мой алгоритм выберет (10 тыс. Строк с более высоким градиентом + x% из оставшихся 490 тыс. Строк, выбранных случайным образом). Предполагая, что x равен 10%, общее количество выбранных строк составляет 59 КБ из 500 КБ, исходя из того, какое значение разделения будет найдено.

> Основное предположение, принятое здесь, состоит в том, что выборки с обучающими экземплярами с небольшими градиентами имеют меньшую ошибку обучения, и она уже хорошо обучена. Чтобы сохранить то же распределение данных, при вычислении прироста информации GOSS вводит постоянный множитель для экземпляров данных с небольшими градиентами. Таким образом, GOSS обеспечивает хороший баланс между уменьшением количества экземпляров данных и сохранением точности для выученных деревьев решений.



Leaf-wise tree growth

Лист с более высоким градиентом / ошибкой используется для дальнейшего роста в ЛГБМ

Как каждая модель относится к категориальным переменным?

CatBoost

CatBoost обладает гибкостью, позволяя задавать индексы категориальных столбцов, чтобы его можно было кодировать как кодирование в одно касание с использованием one_hot_max_size (используйте кодирование в одно касание для всех функций с числом различных значений, меньшим или равным данному значению параметра).

Если вы ничего не передаете в аргументе cat_features, CatBoost будет обрабатывать все столбцы как числовые переменные.

> **Примечание. Если в cat_features не указан столбец со строковыми значениями, CatBoost выдает ошибку. Кроме того, столбец с типом int по умолчанию будет считаться числовым по умолчанию, его необходимо указать в cat_features, чтобы алгоритм воспринимал его как категориальный.**

```
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)
```

Для оставшихся категориальных столбцов, которые имеют уникальное количество категорий, превышающее one_hot_max_size, CatBoost использует эффективный метод кодирования, который аналогичен среднему кодированию, но уменьшает переопределение. Процесс идет так:
1. Перестановка набора входных наблюдений в случайном порядке. Несколько случайных перестановок генерируются
2. Преобразование значения метки из числа с плавающей запятой или категории в целое число
3 Все значения категориальных объектов преобразуются в числовые значения по следующей формуле:

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

Где,**CountInClass**сколько раз значение метки было равно «1» для объектов с текущим значением категориального признака
предшествующийявляется предварительным значением для числителя. Это определяется начальными параметрами.**Общее количество**общее количество объектов (до текущего), которые имеют значение категориального признака, совпадающее с текущим.
Математически это можно представить с помощью приведенного ниже уравнения:

Let $\sigma = (\sigma_1, \ldots, \sigma_n)$ be the permutation, then $x_{\sigma_p,k}$ is substituted with

$$\frac{\sum_{j=1}^{p-1}[x_{\sigma_j,k} = x_{\sigma_p,k}]Y_{\sigma_j} + a \cdot P}{\sum_{j=1}^{p-1}[x_{\sigma_j,k} = x_{\sigma_p,k}] + a}, \qquad (1)$$

LightGBM

Как и в CatBoost, LightGBM также может обрабатывать категориальные функции, вводя имена функций. Он не конвертируется в одноразовое кодирование и намного быстрее, чем одноразовое кодирование. LGBM использует специальный алгоритм, чтобы найти значение разделения категориальных признаков [Ссылка].

Specific feature names and categorical features:

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

Примечание. Перед построением набора данных для LGBM вы должны преобразовать свои категориальные функции в тип int. Он не принимает строковые значения, даже если вы передаете его через параметр categoryorical_feature.

XGBoost

В отличие от CatBoost или LGBM, XGBoost не может обрабатывать категориальные функции сам по себе, он принимает только числовые значения, подобные случайному лесу. Поэтому перед подачей категориальных данных в XGBoost необходимо выполнить различные кодировки, такие как кодирование меток, среднее кодирование или однократное кодирование.

Сходство в гиперпараметрах

Все эти модели имеют множество параметров для настройки, но мы рассмотрим только самые важные. Ниже приведен список этих параметров в соответствии с их функциями и их аналогами в разных моделях.

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

Реализация на наборе данных

Я использую KaggleDatasetзадержек рейсов на 2015 год, поскольку он имеет как категориальные, так и числовые характеристики. Примерно с 5 миллионами строк этот набор данных будет полезен для оценки производительности с точки зрения как скорости, так и точности настроенных моделей для каждого типа усиления. Я буду использовать 10% подмножество этих данных ~ 500 тыс. Строк. Ниже приведены функции, используемые для моделирования:

- **MONTH, DAY, DAY_OF_WEEK**: тип данных int
- **AIRLINE и FLIGHT_NUMBER**: тип данных int
- **ORIGIN_AIRPORT**а также**DESTINATION_AIRPORT:**строка типа данных
- **ВРЕМЯ ОТПРАВЛЕНИЯ:**тип данных float
- **ARRIVAL_DELAY**: это будет цель и преобразуется в логическую переменную, указывающую задержку более 10 минут
- **DISTANCE и AIR_TIME**: тип данных float

```
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)
```

Data Preparation hosted with ❤ by GitHub view raw

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)
```

XGBoost hosted with ❤ by GitHub view raw

Легкий ГБМ

```
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)

```

Light GBM hosted with ❤ by GitHub view raw

CatBoost

При настройке параметров для CatBoost сложно передать индексы для категориальных функций. Поэтому я настроил параметры, не передавая категориальные признаки, и оценил две модели - одну с другими и без категориальных характеристик. Я отдельно настроил one_hot_max_size, потому что он не влияет на другие параметры.

```
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= 0.1
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)
```

CatBoost hosted with ❤ by GitHub view raw

Полученные результаты

	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	

Конечные заметки

Для оценки модели мы должны посмотреть на производительность модели с точки зрения скорости и точности.

Помня об этом, CatBoost становится победителем с максимальной точностью на тестовом наборе (0,816), минимальным переоснащением (точность поезда и тестирование близки) и минимальным временем прогнозирования и временем настройки. Но это произошло только потому, что мы рассмотрели категориальные переменные и настроили one_hot_max_size. Если мы не воспользуемся этими функциями CatBoost, он окажется худшим с точностью 0,752. Следовательно, мы узнали, что CatBoost работает хорошо только тогда, когда в данных есть категориальные переменные, и мы должным образом настроили их.

Нашим следующим исполнителем был XGBoost, который в целом работает хорошо. Точность была довольно близка к CatBoost даже после игнорирования того факта, что в данных есть категориальные переменные, которые мы преобразовали в числовые значения для их потребления. Однако единственная проблема с XGBoost заключается в том, что он работает слишком медленно. Было особенно неприятно настраивать его параметры (мне потребовалось 6 часов, чтобы запустить GridSearchCV - очень плохая идея!). Лучший способ - настроить параметры отдельно, а не использовать GridSearchCV. Проверьте этоблогпост, чтобы понять, как правильно настроить параметры.

Наконец, последнее место занимает Light GBM. Здесь важно отметить, что при использовании cat_features он работал плохо с точки зрения как скорости, так и точности. Я полагаю, что причина, по которой он работал плохо, заключалась в том, что он использовал какое-то модифицированное среднее кодирование для категориальных данных, что вызвало переоснащение (точность поезда довольно высока - 0,999 по сравнению с точностью теста). Однако, если мы используем его обычно как XGBoost, он может достичь аналогичной (если не более высокой) точности с гораздо более высокой скоростью по сравнению с XGBoost
(ЛГБМ - 0,785, XGBoost - 0,789).

Наконец, я должен сказать, что эти наблюдения верны для этого конкретного набора данных и могут или не могут оставаться действительными для других наборов данных. Однако в целом верно то, что XGBoost работает медленнее, чем два других алгоритма.

Так какой из них ваш любимый? Пожалуйста, прокомментируйте с причинами
Будем благодарны за любые отзывы и предложения по улучшению!

Проверьте мои другие блогиВот!

LinkedIn:www.linkedin.com/in/alvira-swalin

Ресурсы

1. http://learningsys.org/nips17/assets/papers/paper_11.pdf
2. https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf
3. https://arxiv.org/pdf/1603.02754.pdf
4. https://github.com/Microsoft/LightGBM
5. https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/

6. https://stats.stackexchange.com/questions/307555/mathematical-differences-between-gbm-xgboost-lightgbm-catboost

Оригинальная статья

- Фреймворки и библиотеки (большая подборка ссылок для разных языков программирования)
- Список бесплатных книг по машинному обучению, доступных для скачивания
- Список блогов и информационных бюллетеней по науке о данных и машинному обучению
- Список (в основном) бесплатных курсов машинного обучения, доступных в Интернете