1 Финальное решение

Полный код метода с результатами выполнения для финального решения можно найти в ./kaggle-ctr/method.ipynb.

1.1 Вычислительные ресурсы

Все считалось на одном MacBook Pro. Контейнер в Docker работал с 8G RAM.

1.2 Метрики

В решении использовались метрики ROC AUC и calibration (с статьи FB). Выборка модели проводилась по ROC AUC, а calibration был для диагностических целей.

1.3 Данные

Использовался train.csv файл с тренировочными данными. Ввиду ограничения одной машины, и довольно сложного pipeline-а со всем столбцами данных, модель разрабатывалась на 40% sample-е всего датасета с разбиением в отношении 80%:10%:10% на train, val и test части. Выборка 40% была проведена равновероятно, в то время как разбиение на части проходило по столбцу id.

Всего в данных 1 target, 13 численных и 26 категориальных признаков. **Все** признаки были использованы для построения модели. Численные признаки были оставлены неизменными (и заполнены 0, если отсутствовали, но тут еще пространство есть). Категориальные же были превращены в численные с помощью MeanTargetEncoding подхода. После первой попытки, прироста с более чем одним таким признаком получать не удавалась (а только ухудшение), но вся проблема была в переобучении. Если какая-то категория встречалась один раз, то MeanTargetEncoding как численный признак назначал ей значение target-а. Обучение на target-е вовсе запрещено в ML. После нескольких часов debug-а возникла такая гипотеза, и решением ей было ограничение на минимальное количество строк в категории, чтобы MTE выставлял ей свое среднее, а не глобальное. Этот параметр **MEAN_TARGET_ENCODER_MIN_EXAMPLES** влиял важным образом на качество модели (ROC AUC), и после ручного перебора его значение установилось на 50. Такой фильтр было легко реализовать подсчитав общие суммы и количества для каждой категории.

В общем-то эта часть решения и являлась секретом получения результата в таблице. Это позволило использовать все 39 признаков для предсказания 1 target-а.

1.4 Модель

Модель была представлена XGBoostEstimator, i.e. gradient boosting with decision trees. Как было описано ранее выборка модели происходила по результату ROC-AUC:

выборка для test-а по максимальному значению метрики на val части, выборка для inference-а (среди других моделей) – по максимальному значению на test. Потерей модели была binary logistic regression. Варьирующими же параметрами модели были colsample_bytree, eta, gamma, max_depth, min_child_weight, subsample, num_round. Начальные значения были равны colsample_bytree=0.9, eta=0.15, gamma=0.9, max_depth=6, min_child_weight=50.0, subsample=0.9, num_round=20. Их оптимизация проходила с помощью пакета hyperopt: сначала пара (num_round, eta), потом (max_depth, min_child_weight) и в конце gamma. Финальные значения colsample_bytree=0.9, eta=0.3, gamma=0.9, max_depth=7, min_child_weight=50.0, subsample=0.9, num_round=100.

Также было экспериментирование с negative downsampling rate, т.к. в датасете всего 298142 положительных (кликов) и 873725 отрицательных примеров. После недолгой оптимизации эффективно все же остался rate=1.0, то есть negative downsampling не использовался (он кажется полезен, когда у вас есть возможность предобработать огромной датасет, а потом из него всего сделать этот downsampling для более быстрой итерации над моделью; т.к. ввиду ограничения RAM получилось обработать только 40% всего датасета, имело смысл все это и использовать).

Достигнутые на test метрики в итоге: {'Calibration Δ ': 0.42160152821518915, 'Area Under ROC': 0.7808397201250101}.

1.5 Inference

Inference происходил с поправкой на negative_downsampling_rate (которого не было в финале :D).

Из интересного, ввиду набора признаков и ограничения RAM пришлось разбить весь inference датасет на 4 равные части (а может просто features столбец отбросить перед конвертацией в rdd?) и append-ить результаты каждого в файл посылки.

2 Предыдущие решения

Финальное решение было объединением hyperopt и gradient boosting notebooks, поэтому имеет смысл лишь вкратце описать решения не из них.

Логистическая регрессия одним махом получала 0.73208 без всяких оптимизаций и считалась на нескольких столбцах. Мой внутренний bias к Gradient boosting был более сильным, поэтому логистическая регрессия дальше не исследовалась.

Логистическая регрессия поверх gradient boosted decision trees хоть и была конкурентной, реализация не была достаточно производительной для обучения на 39 признаках и на большой выборке датасета.