Дата последней сборки документа:
27 декабря 2021 г. в 08:50

Заметки по машинному обучению и анализу данных. Том 2

Подвойский А.О.

Здесь приводятся заметки по некоторым вопросам, касающимся машинного обучения, анализа данных, программирования на языках `Python`, `R` и прочим сопряженным вопросам так или иначе, затрагивающим работу с данными.

Краткое содержание

Содержание

1. Приемы работы с Catboost

Онлайн документация пакета https://catboost.ai/en/docs/concepts/python-reference_catboostregre

1.1. Установка CatBoost

Установить пакет можно с помощью менеджера `conda` (или с помощью `pip`)

```
$ conda config --show channels
# если канала conda-forge нет в списке, то следует его добавить
$ conda config --add channels conda-forge
$ conda install catboost
$ pip install catboost
```

1.2. Ключевые особенности пакета

1.3. Параметры

Ознакомится с описанием параметров можно здесь https://catboost.ai/en/docs/references/training-parameters/

Общие параметры:

- `loss_function` (objective) – функция потерь, которая используется на шаге обучения модели.
- `iterations` – максимальное число деревьев в ансамбле,
- `learning_rate` – темп обучения,
- `l2_leaf_reg` – коэффициент при члене L_2-регуляризации,
- `bagging_temperature` – задает настройки Байесовского бутстрапа

1.4. Классификатор CatBoostClassfier

Класс `CatBoostClassifier`

```
class CatBoostClassifier(
    iterations=None,
    learning_rate=None,
    depth=None,
    l2_leaf_reg=None,
    model_size_reg=None,
    rsm=None,
    loss_function=None,
    border_count=None,
    feature_border_type=None,
    per_float_feature_quantization=None,
    input_borders=None,
```

```
    output_borders=None,
    fold_permutation_block=None,
    od_pval=None,
    od_wait=None,
    od_type=None,
    nan_mode=None,
    counter_calc_method=None,
    leaf_estimation_iterations=None,
    leaf_estimation_method=None,
    thread_count=None,
    random_seed=None,
    use_best_model=None,
    verbose=None,
    logging_level=None,
    metric_period=None,
    ctr_leaf_count_limit=None,
    store_all_simple_ctr=None,
    max_ctr_complexity=None,
    has_time=None,
    allow_const_label=None,
    classes_count=None,
    class_weights=None,
    one_hot_max_size=None,
    random_strength=None,
    name=None,
    ignored_features=None,
    train_dir=None,
    custom_loss=None,
    custom_metric=None,
    eval_metric=None,
    bagging_temperature=None,
    save_snapshot=None,
    snapshot_file=None,
    snapshot_interval=None,
    fold_len_multiplier=None,
    used_ram_limit=None,
    gpu_ram_part=None,
    allow_writing_files=None,
    final_ctr_computation_mode=None,
    approx_on_full_history=None,
    boosting_type=None,
    simple_ctr=None,
    combinations_ctr=None,
    per_feature_ctr=None,
    task_type=None,
    device_config=None,
    devices=None,
    bootstrap_type=None,
    subsample=None,
    sampling_unit=None,
    dev_score_calc_obj_block_size=None,
    max_depth=None,
    n_estimators=None,
    num_boost_round=None,
    num_trees=None,
    colsample_bylevel=None,
    random_state=None,
    reg_lambda=None,
    objective=None,
    eta=None,
```

```
    max_bin=None,
    scale_pos_weight=None,
    gpu_cat_features_storage=None,
    data_partition=None
    metadata=None,
    early_stopping_rounds=None,
    cat_features=None,
    grow_policy=None,
    min_data_in_leaf=None,
    min_child_samples=None,
    max_leaves=None,
    num_leaves=None,
    score_function=None,
    leaf_estimation_backtracking=None,
    ctr_history_unit=None,
    monotone_constraints=None,
    feature_weights=None,
    penalties_coefficient=None,
    first_feature_use_penalties=None,
    model_shrink_rate=None,
    model_shrink_mode=None,
    langevin=None,
    diffusion_temperature=None,
    posterior_sampling=None,
    boost_from_average=None,
    text_features=None,
    tokenizers=None,
    dictionaries=None,
    feature_calcers=None,
    text_processing=None
)
```

LogLoss применяется для задач бинарной классификации (когда целевей вектор содиржит только два уникальных значения или когда параметр `target_border is not None`).

MultiClass используется в задачах мультиклассовой классификации (когда целевой вектор содержит более 2 уникальных значений или параметр `border_count is None`)

1.5. Регрессор CatBoostRegressor

Помимо `iterations` и `learning_rate` у CatBoost 5 важнейших гиперпараметров:

- `max_depth`: мак,
- `l2_leaf_reg`,
- `border_count`,
- `random_strength`,
- `bagging_temperature`.

1.6. Функции потерь и метрики качества

1.6.1. Для классификации

Для мультиклассификации https://catboost.ai/en/docs/concepts/loss-functions-multiclassificat

Функции потерь

- LogLoss

$$-\frac{\sum_{i=1}^{N} w_i\big(c_i \log p_i + (1-c_i)\log(1-p_i)\big)}{\sum_{i=1}^{N} w_i},$$

- CrossEntropy

$$-\frac{\sum_{i=1}^{N} w_i\big(t_i \log p_i + (1-t_i)\log(1-p_i)\big)}{\sum_{i=1}^{N} w_i},$$

Метрики качества

- Precision (точность),
- Recall (полнота),
- F1 (гармоническое среднее),
- BalancedAccuracy

$$\frac{1}{2}\Big(\frac{TP}{T} + \frac{TN}{N}\Big),$$

- BalancedErrorRate

$$\frac{1}{2}\Big(\frac{FP}{TN+FP} + \frac{FN}{FN+TP}\Big),$$

- AUC,
- BrierScore,
- HingeLoss,
- HammingLoss

$$\frac{\sum_{i=1}^{N} w_i[[p_i > 0.5] == t_i]}{\sum_{i=1}^{N} w_i},$$

- Kappa

$$1 - \frac{1 - Accuracy}{1 - RAccuracy},$$
$$RAccuracy = \frac{(TN+FP)(TN+FN) + (FN+TP)(FP+TP)}{\Big(\sum_{i=1}^{N} w_i\Big)^2}$$

- LogLikelihoodOfPrediction.

1.6.2. Для регрессии

Метрики качества, которые могут играть роль функции потерь

- MultiRMSE (в случае мультирегрессии)

$$\left(\frac{\sum_{i=1}^{N}\sum_{d=1}^{dim}(a_{i,d}-t_{i,d})^2 w_i}{\sum_{i=1}^{N} w_i}\right)^{1/2}$$

- MAE

$$\frac{\sum_{i=1}^{N} w_i|a_i-t_i|}{\sum_{i=1}^{N} w_i},$$

- MAPE

$$\frac{\sum_{i=1}^{N} w_i\frac{|a_i-t_i|}{\max(1,|t_i|)}}{\sum_{i=1}^{N} w_i}$$

- Poisson

$$\frac{\sum_{i=1}^{N} w_i(e^{a_i}-a_i t_i)}{\sum_{i=1}^{N} w_i},$$

- Quantile (большие значения α сильнее штрафуют за заниженные прогнозы)

$$\frac{\sum_{i=1}^{N}\Big(\alpha-1[t_i\leqslant a_i]\Big)(t_i-a_i)w_i}{\sum_{i=1}^{N} w_i},$$

- RMSE

$$\left(\frac{\sum_{i=1}^{N}(a_i-t_i)^2 w_i}{\sum_{i=1}^{N} w_i}\right)^{1/2}$$

- LogLinQuantile,
- Lq

$$\frac{\sum_{i=1}^{N}|a_i-t_i|^q w_i}{\sum_{i=1}^{N} w_i}$$

- Huber

$$L(t,a) = \sum_{i=0}^{N} l(t_i, a_i) \cdot w_i, \quad l(t,a) = \begin{cases} \frac{1}{2}(t-a)^2, & |t-a| \leqslant \delta, \\ \delta|t-a| - \frac{1}{2}\delta^2, & |t-a| > \delta. \end{cases}$$

- Excpectile

$$\frac{\sum\limits_{i=1}^{N} |\alpha - 1[t_i \leqslant a_i]|(t_i - a_i)^2 w_i}{\sum\limits_{i=1}^{N} w_i}$$

- Tweedie

$$\frac{\sum\limits_{i=1}^{N} \Big(\frac{e^{a_i(2-\lambda)}}{2-\lambda} - t_i \frac{e^{a_i(1-\lambda)}}{1-\lambda} \Big) w_i}{\sum\limits_{i=1}^{N} w_i},$$

где λ – значение параметра дисперсии мощности,

Метрики качества

- SMAPE

$$\frac{100 \sum\limits_{i=1}^{N} \frac{w_i |a_i - t_i|}{(|t_i| + |a_i|)/2}}{\sum\limits_{i=1}^{N} w_i}$$

- R2 (коэффициент детерминации)

$$1 - \frac{\sum\limits_{i=1}^{N} w_i (a_i - t_i)^2}{\sum\limits_{i=1}^{N} w_i (\bar{t} - t_i)^2}.$$

- MSLE (среднеквадратическая логарифмическая ошибка)

$$\frac{\sum\limits_{i=1}^{N} w_i \big(\ln(1+t_i) - \ln(1+a_i) \big)^2}{\sum\limits_{i=1}^{N} w_i}$$

- MedianAbsoluteError

$$median(|t_1 - a_1|, \ldots, |t_N - a_N|)$$

2. Приемы работы с библиотеками Gym и Ecole

2.1. Gym

Функция окружения (environment) `step` возвращает четыре значения:

- `observation` (object): это объект, специфичный для окружающей среды и представляющий результат наблюдения за этой средой (например, состояние доски в настольной игре),
- `reward` (float): вознаграждение, полученное за предыдущее действие. Масштаб варьируется в зависимости от среды, но цель всегда в том, чтобы сделать суммарное вознаграждение как можно больше,
- `done` (boolean): флаг завершения эпизода. Многие (но не все) задачи разделены на четко определенные эпизоды, и `done = True` указывает на то, что эпизод завершился (например, мы потеряли последнюю жизнь в игре),
- `info` (dict): диагонстическая информация, полезная для отладки.

Это просто реализация классического цикла «агент – среда». На каждом шаге агент совершает то или иное действие и среда возвращает наблюдения (observation) и вознаграждение (reward).

Процесс запускается вызовом функции `reset()`, которая возвращает первое приближение `observation`.

```
import gym
env = gym.make('CartPole-v0')
for i_episode in range(20):
    observation = env.reset()
    for t in range(100):
        env.render()
        print(observation)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
env.close()
```

В этом примере мы отбирали случайные действия из пространства действий среды. Каждая среда поставляется с атрибутами `action_space` и `observation_space`. Эти атрибуты имеют тип `Space` и описывают формат допустимых действий и наблюдений

```
import gym

env = gym.make("CartPole-v0")
print(env.action_space)  # Discrete(2)

print(env.observation_space)  # Box([-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e
    +38], [4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38], (4,), float32)
```

Пространство `Descrete` описывает фиксированный диапазон неотрицательных чисел, так что в данном случае допустимыми действиями будет 0 или 1. Пространство `Box` представляет n-мерный ящик, так что в данном случае допустимыми наблюдениями будут 4-мерные массивы.

2.2. Ecole

Полезный ресурс о специальных приемах работы с задачами линейного программирования в частично-целочисленного постановке https://www.gams.com/37/docs/UG_LanguageFeatures.html?search=sos1

Полезный ресурс по математической оптимизации https://scipbook.readthedocs.io/en/latest/

2.2.1. Observations

Класс `ecole.observation.NodeBipartiteObs`: двудольный граф наблюдений для узлов branch-and-bound дерева. Оптимизационная задача представляется в виде гетерогенного двудольного графа. Между переменной и ограничением будет существовать ребро, если переменная присутствует в ограничении с ненулевым коэффициентом.

Метод `reset()` в Ecole принимает в качестве аргумента экземпляр проблемы.

3. Отбор признаков с библиотекой BoostARoota

BoostARoota https://github.com/chasedehan/BoostARoota – алгоритм отобора признаков на базе экстримального градиентного бустинга в реализации XGBoost. Алгоритм требует гораздо меньших затрат времени на выполнение. Перед применением необходимо выполнить дамми-кодирование, поскольку базовая модель работает только с количественными признаками.

Отбор признаков выполняется на обучающем поднаборе данных, поэтому предполагается, что массив меток и массив признаков *обучающие*, а для проверки качества модели отбора признаков есть независимая, *тестовая* выборка. Кроме того, если необходимо выбрать оптимальные значения гиперпараметров модели отбора признаков (например, значения гиперпараметров `cutoff`, `iters` и `delta`), то понадобиться еще *проверочная* выборка.

4. HDI

Highest Density Interval (HDI) – интервал высокой плотности – показывает какие точки распределения наиболее достоверны/правдоподобны и охватывают большую часть распределения. Каждая точка внутри интервала имеет более высокую *достоверность*, чем любая точка вне интервала.

5. Площадь по ROC-кривой

Построение ROC-кривой происходит следующим образом (рис. 1):

1. Сначала сортируем все наблюдения по убыванию спрогнозированной вероятности положительного класса,
2. Берем единичный квадрат на координатной плоскости. Значения оси абсцисс будут значениями 1 - специфичности (цена деления оси задается значением 1/neg), а значения оси ординат будут значениями чувствительности (цена деления оси задается значением 1/pos). При этом pos — это количество наблюдений положительного класса, а neg — количество наблюдений отрицательного класса,
3. Задаем точку с координатами (0, 0) и для каждого отсортированного наблюдения x:
    - если x принадлежит положительному классу, двигаемся на 1/pos вверх,
    - если x принадлежит отрицательному классу, двигаемся на 1/neg вправо.

Значение вероятности положительного класса, при котором ROC-кривая находится на минимальном расстоянии от верхнего левого угла – точки с координатами (0, 1), дает наибольшую правильность классификации. В данному случае (рис. 2) будет 0.72.

Спрогнозированные вероятности положительного класса, отсортированные по убыванию

№	фактический класс	спрогнозированная вероятность положительного класса
20	P	0,92
19	P	0,9
18	P	0,88
12	N	0,85
17	P	0,82
16	P	0,79
11	N	0,75
15	P	0,73
14	P	0,72
10	N	0,7
9	N	0,6
8	N	0,59
7	N	0,58
6	N	0,53
13	P	0,52
5	N	0,4
4	N	0,33
3	N	0,32
2	N	0,24
1	N	0,18

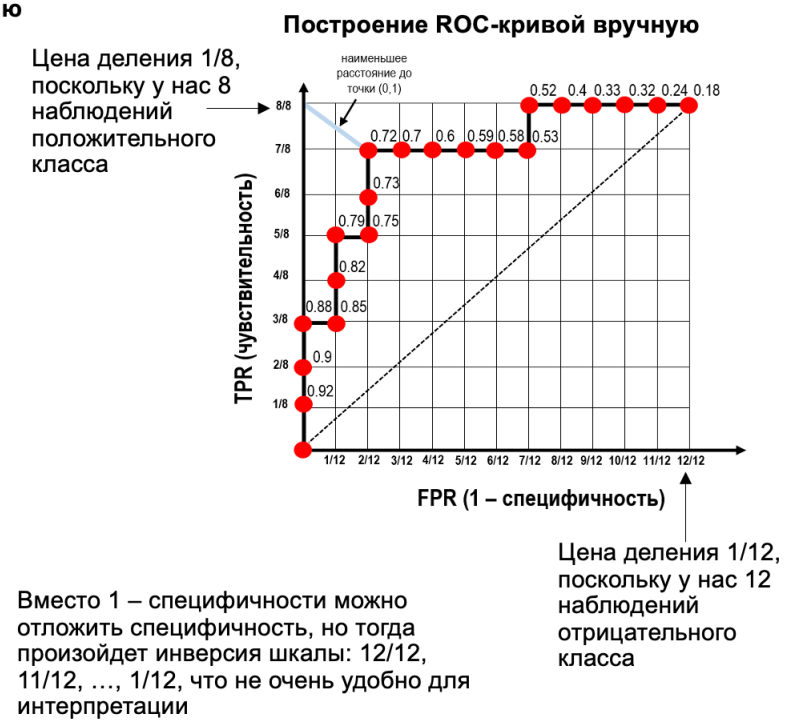

Рис. 1. Построение ROC-кривой

Площадь под ROC-кривой (ROC-AUC) можно интерпретировать как вероятность события, состоящего в том, что классификатор присвоит более высокий ранг (например, вероятность) случайно выбранному экземпляру положительного класса, чем случайно выбранному экземпляру отрицательного класса (если не рассматривать вариант равенства значений рангов).

Замечание

На ROC-кривые не влияет баланс классов (при достаточном объеме выборки) и они могут чрезмерно оптимистично оценивать качество работы алгоритма в случае дисбалансов. Лучше пользоваться гармоническим средним или PR-кривыми

Однако недостаток такой интепретации заключается в том, что мы пренебрегаем часто встречающейся ситуацией равенства вероятностей. Поэтому правильнее будет сказать, что ROC-AUC равен доле пар вида (экземпляр положительного класса, экземпляр отрицательного класса), которые алгоритм верно упорядочил в соответствии с формулой

$$\frac{\sum\limits_{i,j=1}^{n_i,n_j} s(x_i, x_j)}{n_i\, n_j}, \quad s(x_i, x_j) = \begin{cases} 1, x_i > x_j, \\ 1/2, x_i = x_j, \\ 0, x_i < x_j, \end{cases} \tag{1}$$

где x_i – ответ алгоритма для положительного экземпляра, x_j – ответ алгоритма для отрицательного экземпляра.

По сути числитель дроби представляет собой сумму количеств j-ых наблюдений отрицательного класса, лежащих ниже каждого i-ого наблюдения положительного класса. Каждое такое количество мы берем по каждому i-ому наблюдению положительного класса в последовательно-

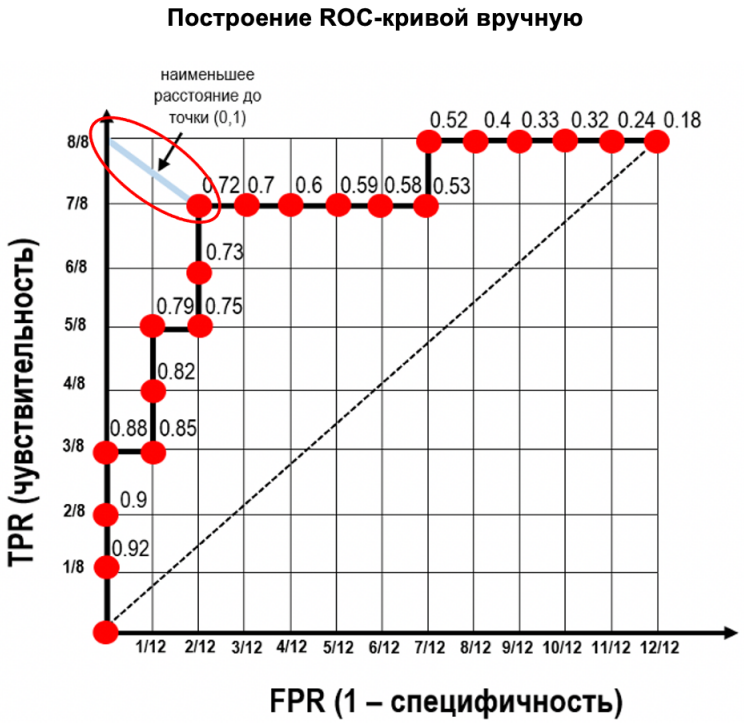

Рис. 2. ROC-кривая. Порог отсечения 0.72

сти, отсортированной по мере убывания вероятности положительного класса. Знаменатель дроби – это произведение количества наблюдений положительного класса и наблюдений отрицательного класса.

Если говорить более точно, мы берем наблюдение положительного класса под номером 20 и каждый раз образовываем пару с наблюдением отрицательного класса (рис. 3), у нас 12 пар, 12 раз наблюдение полжительного класса под номером 20 было проранжировано выше наблюдений отрицательного класса 12, 11, 10 и т.д. Записываем число 12 напротив наблюдения 20.

Разные модели нельзя сравнивать только по ROC-AUC. ROC-AUC оценивает разные классификатор, используя метрику, которая сама зависит от классификатора. То есть ROC-AUC оценивает разные классификаторы, используя разные метрики.

Замечание

Если часть ROC-кривой лежит ниже диагональной линии, а часть – выше, то это означает, что классы не являются линейно-сепарабельными, а при этом используется линейная модель

При одинаковой ROC-AUC у разных моделей (соответственно с разными ROC-кривыми) будет разное распределение стоимостей ошибочной классификации. Проще говоря, мы можем вычислить ROC-AUC для классификатора A и получить 0.7, а затем вычислить ROC-AUC для второго классификатора и снова получить 0.7, но это не обязательно означает, что у них одна и та же эффективность.

6. Приемы работы с Gurobi

Полезный ресурс https://www.gams.com/latest/docs/S_GUROBI.html#GUROBI_GAMS_GUROBI_LOG_FILE

Чтобы запустить Gurobi в интерактвином режиме, следует в командной оболочке набрать `gurobi`

Сессия GUROBI

```
gurobi> m = read("./ikp_milp_problem.lp")
gurobi> m.optimize()
gurobi> vars = m.getVars()
gurobi> help(m)
# вывести 2-картежи целочисленных переменных с отличным от нуля значением
gurobi> [(var.varName, var.x) for var in vars if (var.x > 0) and (var.vType == "I")]
gurobi> m.write("res.sol")  # записать решение
```

Отсортированные спрогнозированные вероятности положительного класса

№	фактический класс	спрогно-зированная вероятность положитель-ного класса	скоринговое правило $S(x_i, x_j) = \begin{cases} 1, x_i > x_j, \\ \frac{1}{2}, x_i = x_j, \\ 0, x_i < x_j \end{cases}$	количество наблюдений отрицательного класса, лежащих ниже соответствующего наблюдения положительного класса
20	P	0,92	0	12
19	P	0,9	0	12
18	P	0,88	0	12
12	N	0,85	1	
17	P	0,82	0	11
16	P	0,79	0	11
11	N	0,75	1	
15	P	0,73	0	10
14	P	0,72	0	10
10	N	0,7	1	
9	N	0,6	1	
8	N	0,59	1	
7	N	0,58	1	
6	N	0,53	1	
13	P	0,52	0	5
5	N	0,4	1	
4	N	0,33	1	
3	N	0,32	1	
2	N	0,24	1	
1	N	0,18	1	

Считаем количество отрицательных ниже каждого наблюдения положи-тельного класса

Рис. 3. Расчет ROC-AUC по формуле (1)

Список иллюстраций

Отсортированные спрогнозированные вероятности положительного класса

случай равенства вероятностей

№	фактический класс	спрогно-зированная вероятность положитель-ного класса	скоринговое правило $S(x_i, x_j) = \begin{cases} 1, x_i > x_j, \\ \frac{1}{2}, x_i = x_j, \\ 0, x_i < x_j \end{cases}$	количество наблюдений отрицательного класса, лежащих ниже соответствующего наблюдения положительного класса
20	P	0,92	0	12
19	P	0,9	0	12
18	P	**0,88**	0,5	11,5
12	N	**0,88**		
17	P	0,82	0	11
16	P	0,79	0	11
11	N	0,75	1	
15	P	0,73	0	10
14	P	0,72	0	10
10	N	0,7	1	
9	N	0,6	1	
8	N	0,59	1	
7	N	0,58	1	
6	N	0,53	1	
13	P	0,52	0	5
5	N	0,4	1	
4	N	0,33	1	
3	N	0,32	1	
2	N	0,24	1	
1	N	0,18	1	

Считаем количество отрицательных ниже каждого наблюдения положи-тельного класса

Рис. 4. Расчет ROC-AUC по формуле (1) для случая равных вероятностей принадлежности экземпляра положительному классу

Список литературы

1. *Лутц М.* Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с.
2. *Бизли Д.* Python. Подробный справочник. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 864 с.