Конспект по теме "Градиентный бустинг"

Ансамбли и бустинг

Ансамбль — набор моделей для решения одной и той же задачи. Сила ансамблей в том, что вместе модели ошибаются в среднем не так сильно, как по отдельности.

Вы уже знакомы с одним типом ансамблевых моделей — случайным лесом. В нём усредняются результаты **базовых моделей**, или **слабых классификаторов** — моделей, из которых состоит ансамбль. Базовые модели в случайном лесе — это решающие деревья.

Второй подход к построению ансамблей — **бустинг**. В нём каждая последующая модель учитывает ошибки предыдущей, а в финальном предсказании — прогнозы, построенные базовыми моделями. Представим итоговую модель, т. е. комбинацию базовых моделей, так:

$$a_N(x) = \sum_{k=1}^{N} \gamma_k b_k(x)$$

где *aN(x)* — предсказание ансамбля, *N* — количество базовых моделей, $b_k(x)$ — предсказание базовой модели, γ_k — вес модели.

Начнём с частного случая бустинга. Например, перед вами задача регрессии. Есть `n` объектов с признаками `x` и правильными ответами `y`. Задача — минимизировать функцию потерь *MSE*:

$$MSE(y, a) = \frac{1}{n} \sum_{i=1}^{n} (a(x_i) - y_i)^2 \longrightarrow \min_{a(x)}$$

Для удобства приравняем веса моделей к единице:

$$\gamma_k = 1, \text{для всех } k = 1, \dots, N$$

Получим:

$$a_N(x) = \sum_{k=1}^{N} b_k(x)$$

Создадим ансамбль последовательных моделей.

Сперва построим базовую модель b_1, решая задачу минимизации:

$$b_1 = \arg\min_b \frac{1}{n} \sum_{i=1}^{n} (b(x_i) - y_i)^2$$

Получим такой ансамбль:

$$a_1(x) = b_1(x)$$

Запишем **остаток** — разность между предсказанием на первом шаге и правильными ответами:

$$e_{1,i} = y_i - b_1(x_i)$$

На втором шаге модель строим уже так:

$$b_2 = \arg\min_b \frac{1}{n} \sum_{i=1}^{n} (b(x_i) - e_{1,i})^2$$

Ансамбль примет такой вид:

$$a_2(x) = \sum_{k=1}^{2} b_k(x) = b_1(x) + b_2(x)$$

Вычислим остаток для каждого объекта *i*:

$$e_{2,i} = y_i - a_2(x_i) = y_i - \sum_{k=1}^{2} b_k(x_i)$$

На каждом следующем шаге алгоритм минимизирует ошибку ансамбля на предыдущем.

Обобщим формулы. На шаге `N-1` остаток вычислим так:

$$e_{N-1,i} = y_i - a_{N-1}(x_i)$$

Сам ансамбль представим как сумму накопленных к этому шагу предсказаний базовых моделей:

$$a_{N-1}(x) = \sum_{k=1}^{N-1} b_k(x)$$

Тогда на *N*-м шаге алгоритм подберёт модель с учётом ошибки ансамбля на шаге *N-1*:

$$b_N(x) = \arg\min_b \frac{1}{n}\sum_{i=1}^{n}(b(x_i) - e_{N-1,i})^2$$

Градиентный бустинг

Каждая базовая модель в бустинге старается «сдвинуть» предсказания прошлого шага в сторону правильных ответов. Так минимизируют функцию потерь. Градиентный спуск помогает делать это эффективнее.

Мы по-прежнему будем минимизировать функцию потерь. Но аргументом, по которому идёт спуск, станут ответы модели! Это и есть **градиентный бустинг** (*gradient boosting*).

К примеру, наша функция потерь — *L(y, a)*, у которой есть производная. Повторим формулу ансамбля:

$$a_N(x) = a_{N-1}(x) + \gamma_N b_N(x)$$

На каждом шаге подберём ответы, которые минимизируют функцию:

$$L(y, a(x)) \longrightarrow \min_a$$

Минимизируем функцию градиентным спуском. Для этого на каждом шаге вычислим антиградиент функции потерь по предсказанию *gN*:

$$g_N(x) = \nabla L(y, a_{N-1}(x) + a)$$

Чтобы сдвинуть предсказания в нужную сторону, следующая базовая модель учится предсказывать *gN*:

$$b_N(x) = \arg\min_b \frac{1}{n} \sum_{i=1}^{N} (b(x_i) - g_N(x_i))^2$$

Вес при *bN* получаем из задачи минимизации, перебирая разные числа:

$$\gamma N = \arg\min_\gamma L(y, a_{N-1}(x) - \gamma b_N(x))$$

Именно коэффициент при базовой модели помогает настроить ансамбль так, чтобы получить наиболее точное предсказание.

Градиентный бустинг годится для разных функций потерь, у которых есть производные. Например, среднеквадратичной в задаче регрессии или логарифмической в задаче бинарной классификации.

Регуляризация градиентного бустинга

Уменьшить переобучение градиента бустинга поможет регуляризация. Если в линейной регрессии снижали веса, то регуляризация для градиентного бустинга — это:

1. уменьшение размера шага;
2. настройка параметров деревьев;
3. рандомизация подвыборок у базовых моделей b_i.

Уменьшим размер шага. Повторим формулу расчёта предсказания на шаге *N*:

$$a_N(x) = a_{N-1}(x) + \gamma_N b_N(x)$$

Если алгоритм делает слишком большие шаги, он быстро запоминает тренировочную выборку, и модель переобучается. Введём коэффициент η, который отвечает за **скорость обучения** (*learning rate*). Он уменьшит размер шага:

$$a_N(x) = a_{N-1}(x) + \eta \times \gamma_N b_N(x)$$

Значение этого коэффициента подбирается экспериментально в диапазоне от 0 до 1. Чем меньше значение η, тем меньше шаг в сторону антиградиента и выше точность ансамбля. Но если скорость обучения слишком низкая, обучение будет идти медленно.

Второй способ регуляризации градиентного бустинга — настройка параметров деревьев. Вы можете ограничить глубину дерева или число объектов в каждом узле, попробовать их разные значения и посмотреть, как это повлияло на результат. Например, для каждого дерева введём ограничение на его глубину — 2. На каждом шаге бустинга нам не придётся создавать сложное дерево, которое точно подстраивается под все остатки. А модель не будет сильно переобучаться.

Третий способ регуляризации — работа с подвыборками. Алгоритм работает не со всей выборкой, а только с подвыборками. Такая версия алгоритма напоминает *SGD* и называется **стохастическим градиентным бустингом** (*stochastic gradient boosting*).

Библиотеки для градиентного бустинга

1. *XGBoost* — популярная на платформе *Kaggle* библиотека градиентного бустинга. Открытое программное обеспечение. Вышло в 2014 году.
2. *LightGBM*. Разработка компании *Microsoft*, быстро и точно обучает градиентный бустинг. Работает с категориальными признаками напрямую. Вышла в 2017 году. Сравнение с *XGBoost*: https://lightgbm.readthedocs.io/en/latest/Experiments.html
3. *CatBoost*. Разработка Яндекса, превосходит другие алгоритмы по метрикам качества. Применяет различные техники кодирования категориальных признаков (*LabelEncoding, One-Hot Encoding*). Вышла в 2017 году. Сравнение с *XGBoost* и *LightGBM*: https://catboost.ai/#benchmark

Сравним библиотеки по двум характеристикам:

Copy of Untitled

Библиотека	Работа с категориальными признаками	Скорость
XGBoost	Нет	Низкая
LightGBM	Да	Высокая
CatBoost	Да	Высокая

Разберём библиотеку *CatBoost*.

Импортируем из библиотеки *CatBoostClassifier* и создадим модель. Поскольку у нас задача классификации, укажем логистическую функцию потерь. Итераций будет 10, чтобы не ждать долго.

```
from catboost import CatBoostClassifier

model = CatBoostClassifier(loss_function="Logloss", iterations=10)
```

Обучим модель методом *fit()*. Помимо целевого признака и признаков, передадим модели категориальные признаки:

```
# cat_features - категориальные признаки

model.fit(features_train, target_train, cat_features=cat_features)
```

Когда итераций много и на каждой информация выводиться не должна, применяют аргумент **verbose**:

```
model = CatBoostClassifier(loss_function="Logloss", iterations=50)
model.fit(features_train, target_train, cat_features=cat_features, verbose=10)
```

Предсказание вычисляется методом *predict()*:

```
pred_valid = model.predict(features_valid)
```