ML BD

Katya Koshchenko

Spring 2020

1 Лекция 7. Hyperparameters

1.1 Задача гиперпараметров.

Какие есть примеры гиперпараметров: в logreg регулязиционные, количество деревьев в GBDT и тд. Какая цель в их оптимизации: 1) улучшить качество мл алгоритмов, 2) вытащить влияние человека из процесса обучения модели, так как на практике обычно их подбирает человек.

Постановка задачи.

- $\mathcal{A}$ — algorithm with N hyperparameters
- Λ_n — domain of n-th hyperparameter
- $\boldsymbol{\Lambda} = \Lambda_1 \times \Lambda_2 \times \ldots \times \Lambda_N$ — configuration space
- $\mathcal{A}_{\boldsymbol{\lambda}}$ — algorithm with hyperparameters $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$

HPO Problem

Diven a dataset $\mathcal{D}$, our goal is to find

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \mathbb{E}_{(\mathcal{D}_{train}, \mathcal{D}_{valid}) \sim \mathcal{D}} \boldsymbol{V}(\mathcal{L}, \mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid}),$$

where $\boldsymbol{V}(\mathcal{L}, \mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$ measure the loss of a model generated by $\mathcal{A}_{\boldsymbol{\lambda}}$ on training data $\mathcal{D}_{train}$ and evaluated on validation data $\mathcal{D}_{valid}$.

Цель: найти такие гиперпараметры, что на них максимизируется матожидание качества работы алгоритма на отложенной выборке. Так как на практике работаем с выборками ограниченного размера, то матожидание заменяется на имперический. V — validation protocol.

GridSearch — простое решение задачи. Для каждого из гиперпараметров в пространстве возможных значений выбираем подмножество, которое будем использовать. Множество точек, которые хотим проверить (trial points) — декартово произвеение отобранных множеств. И, чтобы найти оптимальные гиперпараметры, для каждой точки из множества trial points будем считать функцию потерь на валидации. В конце возьмем точку, дающую лучшие результаты.

Как сравнивают разные алгоритмы поиска гиперпараметров, что говорит о хорошести алгоритма? Главная метрика — сколько нужно было сделать запусков (сколько точек проверили) и сколько времени заняло.

1.2 Проблемы при оптимизации гиперпараметров.

- Обучение модели может занимать очень долго.
- Пространство большое, высокой размерности, сложное, так как декартово произведение очень его раздуевает.
- В пространстве может содержаться обусловленность (conditionality). Некоторые гиперпараметры могут быть использованы только если используются уже другие. Пример: обучаем в нейронной сети и можем не только размерность, но и тип скрытого слоя определить. И тогда от гиперпараметра 'полносвязность слоя' зависит 'размерность слоя'.
- Когда обучаем алгоритм, то можем посчитать производную лосса по параметрам. В гиперпараметрах так сделать не можем, и задача их оптимизации обычно black box, ничего про внутренне устройство алгоритма, обучающего параметры, особо не знаем.

Пример: xgboost, обучающий объявления, обучается 10 часов примерно (данные за последние две недели). Спейс огроменный, в декартовом произведении около 111 миллионов точек. Получаем проклятие размерности.

Как обычно с этим огромным временем борятся. Пусть validation protocol = holdout + cross-validation.

- Используется только подмножество датасета.
- Для итерационных алгоритмов смотрят работу на первых итерациях. Если видим, что плохая конфигурация, то прерываем обучение
- Делают прокси модели, в которых качество сильно связано с качеством основной. Смотрим на качество маленькой и решаем, на каких конфигурациях обучать большую.

Но это делают, когда оптимизируют вручную. А мы хотим это дело автоматизировать.

1.3 Random Search

Random search — model free black box optimization. Есть алгоритм, просто берем и запускаем его на выбранных конфигурациях, это model free. Model based — есть аппроксимация алгоритма обучения, но дешевая.

Как работает Random Search. Основное отличие с гридом — есть большое пространство гиперпараметров, оттуда с равной вероятностью берем и сэмплим какие-то конфигурации. В самом простом случае распределение над значениями каждого из гиперпараметров равномерное. Но вообще ничего не мешает самим задавать распределения гиперпараметров.

Почему это работает. Пусть у нас не больше B раз можем запустить алогоритм (по ресурсам), N — количество гиперпараметров. В гриде мы можем оценить для каждого гиперпараметра не больше $B^(1/N)$ значений (так как декартово произведение). А в случайном поиске B разных, то есть покрытие лучше, но вроде это не гарантия того, что лучше сам алгоритм. На практике чаще всего только небольшое количество гиперпараметров на самом деле имеет значение. В гридсерче для важных параметров рассмотрим только какое-то маленькое количество значений. А так как он важный, то хочется больше точек посмотреть. А в случае с рэндомсерчем покроем больший спектр значений и лучше узнаем о том, как алгоритм зависит от этого гиперпараметра.

Как вообще понять, важен гиперпараметр или нет. Хорошо, если есть априорное знание для старых алгоритмов, но оно есть не всегда. Потом как-то об этом поговорим. Итого, почему обычно грид работает хуже рэндома: 1) маленькое количество гиперпараметров из множества важно, 2) в рамках разных датасетов разные гиперпараметры имеют разную важность.

1.4 Bayesian optimization.

Байесовская оптимизация лучше рэндома. Какая у нас в общем случае задача: максимизация функции, аргументы которой лежат в N-мерном вещественном пространстве. Причем про нее ничего не знаем, а значит не посчитать градиент. Используем вместо этого Sequential Model Optimization.

$$\max_{\mathbf{x}\in\Lambda\subset\mathbb{R}^N} f(\mathbf{x})$$

- $f\colon \Lambda \to \mathbb{R}$ — black-box true objective is costly to evaluate

Algorithm 2.2 SMBO(f, M_0, T, S)

$\mathcal{D}_0 \leftarrow \emptyset$ ▷ Observations
for $t = 1, \ldots, T$ **do**
  $\mathbf{x}^* \leftarrow \arg\min_x S(x, M_{t-1})$
  Evaluate $f(\mathbf{x}^*)$ ▷ Expensive step
  $\mathcal{D}_{1:t} \leftarrow \mathcal{D}_{1:t-1} \cup \{(\mathbf{x}^*, f(\mathbf{x}^*))\}$
  Fit a new model M_t to $\mathcal{D}_{1:t}$
return $\mathcal{D}_{1:t}$ ▷ Observation history

Есть пустое мноество наблюдений. M_0 — аппроксимация функции f, более дешевая. T — колчиество итерацй для поиска глобального максимума. S — трансформация модели (считаем, что identical). Вот взяли M_0, выбираем точку, в которой хотим считать значение целевой функции ($argmin[S]$). Считаем знчение и добавляем точку и значений в ней в наше множество наблюдений. После этого можем аппроксимирующую M_0 как-то обновить.

Теперь байесовская оптимизации. В качестве аппроксимирующей модели используется гауссовский процесс. Потому что можем считать апостериорное через значение функции в определенных точках (короче используем теорему Байеса). Что вообще знаем про гауссовский процесс. Это какое-то обобщение нормального распределения с функциями матожидания и ковариации. В случае гауссовского процесса получаем распределение функции в точке. И еще мно-

жество точек сэмпленных из процесса распределено по многомерному нормальному распределению. Дальше для простоты считаем, что матожидание нулевое, а ковариация $k(x_i, x_j) = \exp(-\frac{1}{2}||x_i - x_j||^2)$. Идея в том, что близкие друг другу точки оказывают друг на друга сильное влияние.

1.4.1 Surrogate Models

Есть априорная функция, хотим наблюдения. Есть гауссовский процесс. Факт: множество значений предыдущих вместе со значением в новой точке ($[\mathbf{f}_{1:t}, f_{t+1}]^T$) имеет нормальное распределение с матожем 0 и матрицей ковариации

$$[[\mathbf{K}, \mathbf{k}], [\mathbf{k}^T, k(x_{t+1}, x_{t+1})]], \mathbf{k} = [k(x_{t+1}, x_1), ..., k(x_{t+1}, x_t)], \mathbf{K} = (k(x_i, x_j))_{i,j} - kernelmatrix$$

И суррогатная (аппроксимирующая) функция в случае байесовской оптимизации — это вероятность получить значение следующей точки, имеющее нормальное распределение

$$P(f_{t+1}|D_{1:t}, x_{t+1}) = N(\mu_t(x_{t+1}), \sigma_t^2(x_{t+1})),$$

$$\mu_t(x_{t+1}) = \mathbf{k}^T\mathbf{K}^{-1}\mathbf{f}_{1:t}$$
$$\sigma_t^2(x_{t+1}) = k(x_{t+1}, x_{t+1}) - \mathbf{k}^T\mathbf{K}^{-1}\mathbf{k}$$

Таким образом, шаг за шагом можем обновлять апостериорное распределение значений функции.

Проблема подхода: вычислительная сложность, так как там есть обращение матриц, а это куб операций от количества текущих наблюдений. Но обычно в байесовой оптимизации, чтобы найти близкое к глобальному оптимиуму значение, нужно сделать мало наблюдений.

В используемой функции оптимизации могут быть гиперпараметы: θ, которая конролирует ширину ядра. В гауссовских все просто, можно найти через Minimum likelihood (mle). Есть множество наблюдений, можем log-likelihood записать, он зависит от ковариации (можем еще и матожидание параметризировать). И просто производную ll считаем по этим параметрам.

1.4.2 Acquisition Functions

Итак, есть сурогатная модель, оптимизирующая нашу функцию целевую. Хотим теперь найти точку, в которой будем считать значение этой целевой. Для этого используем фукнцию выгоды, и ищем ее аргмакс. Высокие значения функции выгоды соотвествуют регионам, в которых может быть маленькая неопределенность. Но знаем, что значение функции скорее всего будет большое (trade-off exploration/exploitation). Пример выгоды: вероятность улучшения

$$PI(x) = P(f(x) >= f(x^+)) = \Theta(\frac{\mu(x) - f(x^+)}{\sigma(x)})$$

$$x^+ = argmax_{x_i \in x_{1:t}} f(x_i)$$

Это exploitation, просто жадно решаем, в какой точке будем считать функцию. Идея в том, что считать значение функции выгоды гораздо проще, чем целевой.

1.4.3 Noise

На практике обычно бывает шум при вычислении значения в одной точке. И байесовская оптимизация обычно позволяет легко добавить этот шум в модель. Скажем, что наблюдаем значение функции с белым шумом распределенным $N(0, \sigma^2_{noise})$. И при этом шум друг на друга влияние не оказывает. Тогда к ковариации просто добавили диагональную матрицу стандартного отклонения шума. Но вот опять у нас есть параметр (отклонение шума). Чтобы его найти, во время обучения гауссовского процесса просто подбираем стандартное отклонение шума, как и раньше подбирали.

1.5 Multi-fidelity Optimization

Это ускорение методов, которые уже рассмотрели. Он находит трейдоф на время обучения алгоритма и качество обучения.

1.5.1 Predictive Termination.

Люди при обучении нейросетки часто смотрят на Learning Curve, и если текущий алгоритм сильно проигрывает хорошему, то просто останаваливаем. Или можем сказать, что качество алгоритма — фукнкция от размера выборки. Обучили на 10%, чекнули качество, добавили данные и тд. И каждый раз смотрим на кривую и решаем, останаваливать или нет.

Вот придумали алгоритм, позволяющий убрать челвоека из этой системы. Когда предсказываем, хотим получить вероятность, что новая пофиченная кривая на каких-то больших итерациях будет не хуже имеющихся наблюдений. Если вероятность превышает пороговое значение, то продолжаем обучение.

1.5.2 Hyperband.

Есть множество конфигураций, которые хотим протестировать. Для каждой точки в нем хотим обучить алгоритм. Сначала каждому из алгоритмов дадим обучаться на небольшом фрагменте датасета. И половину алгоритмов, которые работали хуже всего, отбрасываем. Оставшимся в два раза увеличили количество данных. Затем опять отбрасываем худшие и тд. Так делаем, пока не останется один алгоритм (один набор гиперпараметров). Проблема: юзер решает, сколько конфигураций параллельно обучает, какой дать изначально бюджет и тд.

1.5.3 Multi-task Bayesian Optimization.

Давайте обучать алгоритм на небольшом подмножестве датасета. Вот обучили несколько конфигураций на небольшом множестве, а несколько в тех же точках, но на всем датасете. И если их качества хорошо скоррелированы, то маленькая может хорошо делать подсказку, какое будет качество у большой.