Машинное обучение, ФКН ВШЭ
Семинар №10
Генерация и отбор признаков. Стекинг. Интерпретация алгоритмов

Решение практической задачи машинного обучения обычно состоит из четырех важных этапов:

1. Визуализация и предобработка данных (включая заполнение пропусков, нормировку, деление выборки)
2. Создание хорошего признакового описания (включая дополнительную нормировку при необходимости)
3. Настройка алгоритмов (включая построение бейзлайна, настройку гиперпараметров алгоритмов)
4. Построение композиции настроенных алгоритмов

В некоторых приложениях важен также 5-й шаг — анализ итогового алгоритма (почему алгоритм делает такие предсказания? удовлетворяет ли алгоритм требованиям?).

О пунктах 1 и 3 мы подробно говорили на предыдущих занятиях. Сегодня мы обсудим шаги 2, 4 и 5.

1 Генерация новых признаков

В курсе мы подробно изучаем несколько семейств параметрических алгоритмов машинного обучения. Все эти алгоритмы могут выучивать зависимости только определенного вида (узкое подмножество всех возможных зависимостей). Кроме того, даже если само семейство алгоритмов достаточно общее, процедура обучения обычно не позволяет найти самый оптимальный алгоритм.

Чтобы расширить множество моделей, можно вручную добавить новые признаки, посчитанные из имеющихся данных (feature exatraction). Тем самым мы как бы помогаем алгоритму, выполняя часть работы за него. Качественный подборе признакового пространства обычно оказывается очень важным для достижения лучшего качества решения задачи, например, в соревнованиях на Kaggle.

Какие признаки стоит добавить, конечно же, зависит от задачи. Частично решению этого вопроса помогает визуализация данных. Ниже мы опишем серию популярных приемов создания новых признаков. Понятно, что это скорее большой пул эвристик, чем конкретные рекомендации, и далеко не все приемы будут хорошо работать на конкретных данных.

§1.1 Вещественные признаки

На основе вещественных признаков можно добавлять следующие признаки:

- нелинейные трансформации признаков (изменение шкалы признака, "выпрямление"зависимости — особенно для линейных моделей);
- агрегация нескольких признаков одной группы в один (например, взятие максимума/медианы по оценкам студента)
- дискретизация признака:
  - по порогу (например, на визуализации гистограммы признака видно два четких множества)
  - кластеризация значений признака
  - округление (например, выделение возрастных групп вместо конкретного значения возраста)

§1.2 Даты и время

- Вычисление дня недели, месяца, года из даты
- Вычисление часа, минуты, секунды из метки времени
- Праздник / выходной
- Специфичные даты для задачи
- Количество времени, прошедшего с какого-то специфичного для задачи события

Природа временных данных такова, что многие последовательности зациклены (например, после часа 23 идет час 0). В некоторых задачах может быть важно это учесть, в то время как использование номинального или категориального признака это не учитывает. Иногда используют циклическое кодирование: располагают элементы последовательности (например, дни недели) на двумерной окружности и используют двумерные координаты каждой точки в качестве кода.

Важный нюанс с датой и временем состоит в том, что их нужно учитывать при разделении выборки на обучение и контроль.

§1.3 Географические признаки

Здесь подразумевается создание новых признаков на основе широты и долготы:

- кластеризация точек
- оценка плотности точек в области
- расстояние до важных для задачи объектов, до центров кластеров
- получение координат по адресу и адреса по кооридинатам (через API карт, например, Google)

§1.4 Категориальные признаки

Помимо стандартного one-hot кодирования, можно использовать разные приемы:

- нумерация категорий + нумерация несколько раз в разном порядке (приемлемо для решающих деревьев)
- хеширование (с коллизиями) + повторение несколько раз
- категория → число объектов в этой категории
- усреднение значений вещественных признаков по каждой категории (например, средний балл студентов разных направлений)
- счетчики (усреднение значений целевого признака по каждой категории, нужно дополнительное деление выборки)

§1.5 Текстовые данные

В этом разделе предполагается, что для каждого объекта задан некоторый текст, и необходимо извлечь признаковое представление фиксированной размерности из этого текста:

- мешок слов, TF-IDF
- мешок n-грамм (учет порядка слов)
- мешок n-букв (устойчивость к опечаткам): двигаемся по символам окошком длины n и считаем количество различных буквосочетаний
- использование скрытых представлений (embeddings):
  - word2vec, doc2vec, ELMO, BERT ...
  - тематическое моделирование

§1.6 Изображения

В этом разделе предполагается, что для каждого объекта задано некоторое изображение, и необходимо извлечь признаковое представление фиксированной размерности из этого изображения:

- самое стандартное сегодня — представления, полученные на некотором слое предобученной нейронной сети
- гистограммы яркости, градиентов — из классического компьютерного зрения

§1.7 Общие методы

Помимо индивидуальной работы с каждым конкретным признаком, можно получать новые представления сразу для всего признакового представления объекта:

- кластеризация объектов и добавление нового признака — кластера объекта
- расстояния до эталонных объектов, до центров кластеров
- выходы алгоритмов как признаки (обучаем простой алгоритм, добавляем его предсказания как признаки и обучаем сложный алгоритм; требуется аккуратное деление выборки)
- методы понижения размерности (PCA, автокодировщики)

2 Методы отбора признаков

В предыдущем разделе мы разобрали, как можно расширить множество признаков, чтобы подсказать модели зависимости, которые она, возможно, сама найти не сможет. Однако очевидно, что среди этих признаков много ненужных, или шумовых — тех, которые могут мешать делать качественные предсказания. Кроме того, чем больше признаков, тем дольше выполняются обучение и предсказание и тем сложнее анализировать модель. Возникает потребность в отборе признаков.

Часто отбор признаков опирается на оценку важностей $R_1, \ldots, R_d$ признаков: каждому признаку нужно присвоить оценку, насколько он влияет на целевую переменную. В некоторых подразделах мы будем говорить только о подсчете важности признаков; понятно, что на основе оценок важности несложно произвести отбор признаков.

Методы отбора признаков делятся на три группы:

- фильтрация (отбор признаков без учета модели);
- методы-обертки (wrapper methods, выбор признаков, дающих лучшее качество для модели);
- отбор с помощью моделей (embedded methods, использование свойств моделей для оценки важностей признаков).

Ниже мы подробнее обсудим все три группы.

§2.1 Методы фильтрации

Эта группа методов использует аппарат статистики для оценки зависимости целевого вектора от признаковых описаний объектов. Проще всего оценить зависимость между двумя одномерными векторами — вектором x_j значений j-го признака и целевым вектором y.

2.1.1 Корреляция

$$R_j = \frac{\sum_{i=1}^{\ell}(x_{ij} - \bar{x}_j)(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{\ell}(x_{ij} - \bar{x}_j)^2 \sum_{i=1}^{\ell}(y_i - \bar{y})^2}}$$

Как известно, корреляция учитывает только линейную связь между величинами (если связь существует и нелинейная, корреляция может оказаться равной 0).

2.1.2 T-score

Для задачи бинарной классификации:

$$R_j = \frac{|\mu_0 - \mu_1|}{\sqrt{\frac{\sigma_0^2}{n_0} + \frac{\sigma_1^2}{n_1}}},$$

где μ_0 / σ_0^2 / n_0 и μ_1 / σ_1^2 / n_1 — средние / дисперсии j-го признака / число объектов в 0 и 1 классах соответственно.

2.1.3 F-score

Обобщение для задачи K-классовой классификации:

$$R_j = \frac{\sum_{k=1}^{K} \frac{n_j}{K-1}(\mu_j - \mu)^2}{\frac{1}{\ell - K}\sum_{k=1}^{K}(n_j - 1)\sigma_j^2},$$

где μ — среднее признака по всей выборке, остальные обозначения аналогичны предыдущему пункту.

Впрочем, оценить зависимость одного вектора от другого можно и без критериев — достаточно построить scatter-графики в осях x_j и y и визуально оценить наличие зависимости (если признаков не очень много). Однако использовать подобные процедуры для нескольких признаков одновременно уже не получится (а на практике зависимости именно многомерные).

§2.2 Методы-обертки

В методах-обертках отбор признаков производится с помощью обучения алгоритма на разных признаковых подпространствах и оценке его качества на контрольных данных. Очевидно, что если перебрать все возможные подмножества признаков, то мы сможем выбрать наилучшее подмножество. Конечно же, на практике это возможно только при совсем маленьком числе признаков (до 5).

В качестве альтернативы полному перебору можно использовать жадные алгоритмы. Например, можно инициализировать множество признаков $F = \emptyset$, и затем итерационно добавлять в F по одному признаку, выбирая на каждой итерации признак, дающий наибольший прирост в качестве. Разработано множество модификаций такого алгоритма: например, можно поочередно то жадно добавлять, то жадно удалять признаки (так мы исследуем больше вариантов). Преимущества и недостатки

такого подхода очевидны: жадные алгоритмы просты в реализации и эффективны, однако решают задачу далеко не оптимально.

Еще одна весьма забавная группа алгоритмов — генетические алгоритмы[1]. В них проводится аналогия с эволюцией: каждый набор признаков — это организм, задаваемый бинарной маской $\beta = \{0,1\}^d$ (1 — признак входит в подмножество, 0 — не входит). Вводят понятия скрещивания и мутации:

- скрещивание $\beta = \beta' \times \beta''$: $\beta_j = \begin{cases} \beta'_j, & \text{с вер. } 0.5 \\ \beta''_j, & \text{иначе} \end{cases}$
- мутация $\beta = \sim \beta'$: $\beta_j = \begin{cases} \beta'_j, & \text{с вер. } 0.5 \\ 1 - \beta'_j & \text{иначе} \end{cases}$

При инициализации алгоритма генерируют m случайных организмов. Далее на каждой итерации (поколении) с помощью мутаций и скрещиваний получают новое множество организмов и выбирают из них m лучших (на каждом подмножестве признаков обучают алгоритм и выбирают лучшие по качеству на контроле).

§2.3 Подсчет важностей признаков на основе моделей

Наконец, можно использовать свойства самих алгоритмов машинного обучения, чтобы оценивать важности признаков и затем на основе этих важностей производит отбор признаков. Подсчет величин R_j важен также для интерпретации обученного алгоритма (подробнее об интерпретации ниже).

2.3.1 Подсчет важности признаков в линейных моделях

В линейных моделях важность признаков определяется интуитивно — по величине веса при данном признаке. Главное, чтобы все признаки были в одной шкале — для этого данные необходимо нормировать. При использовании L_1-регуляризации производится автоматический отбор признаков. Преимущество такого подхода в простоте, недостаток — в том, что оцениваются только линейные зависимости.

2.3.2 Подсчет важности признаков в решающих деревьях и их композициях

В отличие от линейных моделей, в решающих деревьях нет конкретного параметра, показывающего важность признака. Приходится придумывать эвристические методы. Важно понимать, что эти методы лишь анализируют уже построенную модель согласно кажущимся разумными принципам, поэтому не стоит 100% им доверять.

Может показаться, что признак тем важнее, чем чаще по нему производится расщепление в вершине решающего дерева. Однако часто такие расщепления настраиваются на шум в данных (например, отделяют по одному объекту в лист) и не отображают реальные зависимости. В то же время, нельзя сделать и противоположное утверждение — о том, что признак тем важнее, чем меньше по нему производится расщепление (шумовые признаки тоже редко выбираются в вершинах).

[1] По материалам Константина Воронцова

Рассмотрим другую эвристику. Вспомним, что при построении вершины решающего дерева признак j и порог t, по которым выборка X делится на X_ℓ и X_r, выбираются согласно критерию

$$Q(j,t) = H(X) - \frac{|X_\ell|}{|X|}H(X_\ell) - \frac{|X_r|}{|X|}H(X_r),$$

где $H(X)$ — критерий информативности, определяемый задачей. Пусть мы уже построили дерево. Логично считать, что признак тем важнее, чем сильнее его использование уменьшало $H(X)$, то есть чем меньше $Q(j,t)$ во всех вершинах, в которых разбиение производилось по этому признаку. Таким образом, для подсчета важности R_j признака j нужно просуммировать $Q(j,t)$ по всем вершинам, в которых разбиение выполнялось по признаку j. Для композиции деревьев суммирование нужно проводить по всем соответствующим вершинам всех деревьев. Такой алгоритм реализован в sklearn (атрибут feature_importances_ решающего дерева или случайного леса).

В языке R, помимо указанного, реализован еще один алгоритм[2]. В этом алгоритме используется интересная и в то же время простая эвристика: если признак важный, то при замене его на случайно генерируемый признак качество решения задачи сильно упадет. Чтобы максимально сохранить признаковое пространство и распределение данных, признак не удаляют и не заменяют константой. Вместо этого можно просто перемешать значения признака во всех объектах обучающей выборки.

Понятно, что смотреть на изменение качества нужно на контроле, а не на обучении. В случайном лесе вместо этого можно использовать out-of-bag оценки. Напомним, что out-of-bag ошибка композиции $a(x) = \sum_{n=1}^{N} b_n(x)$, построенной с помощью бэггинга, вычисляется по формуле (подробнее см. конспект лекции по бэггингу):

$$\text{OOB} = \sum_{i=1}^{\ell} L\left(y_i, \frac{1}{\sum_{n=1}^{N}[x_i \notin X_n]}\sum_{n=1}^{N}[x_i \notin X_n]b_n(x_i)\right)$$

Итоговый алгоритм подсчета важностей для случайного леса выглядит так:

1. Вычислить OOB для случайного леса, обученного по исходной обучающей выборке
2. Для каждого признака j:
   (а) Перемешать значения признака по всем объектам обучающей выборки
   (б) Вычислить OOB_j случайного леса, обученного по измененной обучающей выборке
   (в) Оценить важности: $R_j = \max(0, \text{OOB} - \text{OOB}_j)$

3 Стекинг

На предыдущих занятиях мы обсудили бустинг и бэггинг - композиционные алгоритмы, позволяющие объединять много слабых алгоритмов в одну сильную композицию. Предположим, что мы уже построили несколько сильных алгоритмов для решения задачи, и хотим их тоже объединить в композицию.

[2]По материалам Александра Дьяконова

Большую популярность для решения этой задачи имеет *стекинг*, в котором прогнозы алгоритмов объявляются новыми признаками, и поверх них обучается ещё один алгоритм (который иногда называют мета-алгоритмом). Стекинг очень популярен в соревнованиях по анализу данных, поскольку позволяет агрегировать разные модели (различные композиции, линейные модели, нейросети и т.д.; иногда в качестве базовых алгоритмов могут выступать результаты градиентного бустинга с разными значениями гиперпараметров).

Допустим, мы независимо обучили N базовых алгоритмов $b_1(x), \ldots, b_N(x)$ на выборке X, и теперь хотим обучить на их прогнозах мета-алгоритм $a(x)$. Самым простым вариантом будет обучить его на этой же выборке:

$$\sum_{i=1}^{\ell} L(y_i, a(b_1(x_i), \ldots, b_N(x_i))) \to \min_a$$

При таком подходе $a(x)$ будет отдавать предпочтение тем базовым алгоритмам, которые сильнее всех подогнались под целевую переменную на обучении (поскольку по их прогнозам лучше всего восстанавливаются истинные ответы). Если среди базовых алгоритмов будет идеально переобученный (то есть запомнивший ответы на всей обучающей выборке), то мета-алгоритму будет выгодно использовать только прогнозы данного переобученного базового алгоритма, поскольку это позволит добиться лучших результатов с точки зрения записанного функционала. При этом такой мета-алгоритм, конечно, будет показывать очень низкое качество на новых данных.

Чтобы избежать таких проблем, следует обучать базовые алгоритмы и мета-алгоритм на разных выборках. Разобьём нашу обучающую выборку на K блоков $X_1, \ldots, X_K$, и обозначим через $b_j^{-k}(x)$ базовый алгоритм $b_j(x)$, обученный по всем блокам, кроме k-го. Тогда функционал для обучения мета-алгоритма можно записать как

$$\sum_{k=1}^{K} \sum_{(x_i, y_i) \in X_k} L\left(y_i, a(b_1^{-k}(x_i), \ldots, b_N^{-k}(x_i))\right) \to \min_a$$

В данном случае при вычислении ошибки мета-алгоритма на объекте x_i используются базовые алгоритмы, которые не видели этот объект при обучении, и поэтому мета-алгоритм не может переобучиться на их прогнозах.

Блендинг. Частным случаем стекинга является блендинг, в котором мета-алгоритм является линейным:

$$a(x) = \sum_{n=1}^{N} w_n b_n(x).$$

Это самый простой способ объединить несколько алгоритмов в композицию. Иногда даже блендинг без обучения весов (то есть вариант с $w_1 = \cdots = w_N = 1/N$) позволяет улучшить качество по сравнению с отдельными базовыми алгоритмами.

4 Интерпретация алгоритмов

В курсе мы подробно обсуждаем, как добиваться хорошего качества решения задачи: имея выборку X, y, построить алгоритм с наименьшей ошибкой. Однако за-

казчику часто важно понимать, как работает алгоритм, почему он делает такие предсказания. Обсудим несколько мотиваций.

Доверие алгоритму. Например, а банках на основе решений, принятых алгоритмом, выполняются финансовые операции, и менеджер, ответственный за эти операции, будет готов использовать алгоритм, только если он понимает, что его решения обоснованы. По этой причине в банках очень часто используют простые линейные алгоритмы :) Другой пример — из области медицины: поскольку цена ошибки может быть очень велика, врачи готовы использовать только интерпретируемые алгоритмы.

Отсутствие дискриминации (fairness). Вновь пример с банком: алгоритм кредитного скоринга не должен учитывать расовую принадлежность (racial bias) заемщика или его пол (gender bias). Между тем, такие зависимости часто могут присутствовать в датасете (исторические данные), на котором обучался алгоритм. Еще один пример: известно, что вектора word2vec содержат gender bias. Если эти вектора использовались при построении системы поиска по резюме для рекрутера, то например по запросу «technical skill» он будет видеть женские резюме в конце ранжированного списка.

Учет контекста. Данные, на которых обучается алгоритм, не отображают всю предметную область. Интерпретация алгоритма позволит оценить, насколько найденные зависимости связаны с реальной жизнью. Если предсказания интерпретируемы, это также говорит о высокой обобщающей способности алгоритма.

Как уже было сказано, полной интерпретируемостью отличаются линейные алгоритмы. Хорошо интерпретируемыми также считаются неглубокие деревья, однако и эта модель является слишком простой. Для более сложных алгоритмов используют методы определения важности признаков, которые мы уже обсудили в предыдущих разделах.

§4.1 LIME

Отдельным направлением исследований является post-hoc анализ уже построенных алгоритмов. Разберем наиболее популярный метод — LIME.

Предположим, что мы обучили алгоритм a на выборке X, y, и хотим интерпретировать его предсказания. Алгоритм LIME выполняет локальную интерпретацию, то есть объясняет предсказания для конкретного объекта. Основная идея состоит в локальном приближении разделяющей поверхности алгоритма другим, более простым алгоритмом b (например, линейным). Итак, чтобы объяснить предсказания $a(x)$ на конкретном объекте x:

1. Сгенерировать выборку точек $x_1, \ldots, x_m$ около объекта x.
2. Обучить интерпретируемый алгоритм b на выборке $\{(x_1, a(x_1)), \ldots, (x_m, a(x_m))\}$.
3. Интерпретировать алгоритм b.

Если мы приближаем разделяющую поверхность линейной функцией b, то по обученным весам можно сказать, какой вклад вносит каждый признак в предсказания в окрестности точки x.

Список литературы

[1] *Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin* (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. // https://arxiv.org/abs/1602.04938