Машинное обучение: kNN, проклятье размерности, ближайшие соседи

Кураленок И.Е.

Яндекс

26 апреля 2012 г.

План

Принцип локальности (совсем не физика)

Близкие точки похожи!

- что значит близки?
  l_q, косинусная мера, Mahalanobis distance, KL-divergence, более хитрые преобразования, Metric Learning etc.
- что значит похожи?
  близкие значения целевой функции, возможности простой аппроксимации, схожее распределение, etc.
- что значит точки?
  точки из learn, центройды классов, прочие прототипные точки.

Алгоритм k-NN

1. Вычислим значения факторов интересующей нас точки.
2. Найдем k ближайших по выбранной метрике точек из learn'a.
3. Агрегируем значения искомой характеристики для найденных точках.

Параметры k-NN

1. Сколько точек выбрать?
   ничего лучше продбора по validate или crossfold validation не придумали
2. Как искать соседей?
   см. вторую часть
3. Способ агрегации.
   голосовалка, усреднение, моделирование распределения

Далее речь пойдет про мульти-классификацию на K классов.

Свойства k-NN

- Очень часто работает!
- Простота реализации и наглядность.
- Ничего не требует на стадии обучения.
- Известная оценка сверху эффективности при условии отсутствия bias'a в learn

$$
\begin{array}{l}
BE = 1 - p(x|k^*) \\
E = \sum_{i=1}^{K} p(x|k)\,(1 - p(x|k)) \\
BE \leq E \leq 2BE
\end{array}
$$

- Часто требователен на фазе решения (!)

Основные способы прототипирования

Может много точек и не надо? Будет быстрее работать, глаже границы.

Как выбрать прототипные точки:

- выбрать рандомно;
- покластеризовать каждый класс (например k-means'ами) и назначить центроиды прототипами;
- выбрать как-нибудь точки и подвигать их подальше от границ классов

Learning Vector Quantization

1. построить вашим любимым способом прототипные точки
2. до сходимости уменьшая ϵ (learning rate):
   1. Случайно, равномерно выберем точку x из learn'a с возвращением
   2. Найдем ближайший прототип

   прототип нужного класса: подвинем его поближе к точке

   $$m_j^{t+1} = m_j^t + \epsilon(x - m_j^t)$$

   "чужой" прототип: подвинем его подальше

   $$m_j^{t+1} = m_j^t - \epsilon(x - m_j^t)$$

Проклятье размерности

Что происходит с расстояниями когда размерность увеличивается?
Проведем простой опыт: будем равномерно выбирать точки из кубика $[0, 1]^n \subset \mathbb{R}$.
Оказывается что при увеличении n:

- Точки все ближе "жмутся" к краю
- Углы между точками выравниваются
- Окрестности все чаще упираются в границы
- Для того, чтобы пространство было плотным надо слишком много точек

$\Rightarrow$ Большая размерность — зло для kNN и не только для него!

Discriminant Adaptive Nearest-Neighbor (DANN)

Идея: а давайте при в расстоянии учитывать локальную топологию в искомой точке

Выберем много ближайших соседей (например m=50):

$$\begin{aligned} T &= m\Sigma = \sum_{i=1}^{m}(x_i - \mu(x))(x_i - \mu(x))^T \\ &= \sum_{k=1}^{K}\sum_{i \in I_k}(x_i - \mu_k(x))(x_i - \mu_k(x))^T \\ &\quad + \sum_{k=1}^{K}(\mu_k(x) - \mu(x))(\mu_k(x) - \mu(x))^T \\ &= W + B \end{aligned}$$

Пересчитаем все расстояния для k-NN:

$$D(x, x_0) = (x - x_0)^T \mathcal{D}(x - x_0)$$
$$\mathcal{D} = W^{-\frac{1}{2}}\left(W^{-\frac{1}{2}}BW^{-\frac{1}{2}} + \epsilon E\right)W^{-\frac{1}{2}}$$

Заметим, что ранги матриц, которые мы используем не более m. Получается очень точный, но крайне медленный метод.

Поиск ближайших соседей

Это область на годовой курс, так что за поллекции мы ничего не успеем :).

- Линейный поиск
- Разбиение пространства
- Чувствительное к локальности хеширование (LSH)
- Кластерный/Пожатый поиск
- Деревья в пространстве меньшей размерности

kd-дерево

Идея: разложим множество по потором у будем искать в бинарное дерево с простыми условиями и конкретными точками в узлах. Будем надеяться на правило треугольника (не подходит для cosine(!)) Некоторые свойства:

- Один из самый простых способов поиска ближайших соседей.
- Работает только в малых размерностях.
- Затратные алгоритмы перестроения (если нужна динамика смотрим R-деревья).

kd-дерево: построение

1. По циклу, или рандомно выбираем ось.
2. Ищем точку, разбивающую множество на как можно более равные части.
3. Работает только в малых размерностях.
4. Повторяем 1-3 для каждого из получившихся подмножеств

Сложность: $O(n \log n)$

kd-дерево: поиск

1. Бежим по бинарному дереву поиска.
2. Когда добежали, сохраняем листовую точку как $\bar{x}$ – текущую лучшую.
3. Бежим обратно вверх по дереву:
   1. если текущая точка лучше то теперь она $\bar{x}$;
   2. сравниваем расстояние от точки-запроса до гиперплоскости текущего уровня
      - пересекает: не повезло, бежим в поддерево
      - не пересекает: повезло, бежим наверх.
4. Повторяем 1-3 для каждого из получившихся подмножеств пока делятся.

Сложность сильно зависит от множества: в лучшем случае $O(\log n)$, в худшем $O(nN^{1-\frac{1}{n}})$ Классический пример проклятия размерности

Немного определений

Рандомизированнный R-ближайший сосед: если есть R-соседи, то алгоритм должен вернуть каждого из них с вероятностью $1 - \delta$.

Р-я c-аппроксимация R-ближайшего соседа $((c, R) - NN)$: если есть R-сосед, то алгоритм должен вернуть хотя бы одного cR-соседа с вероятностью $1 - \delta$.

Locality-Sensitive Hash Functions

$\mathcal{H} = \{h | h : \mathbb{R}^n -> \mathbb{Z}\}$ – (R, cR, p_1, p_2):

$$\|p - q\| \leq R \Rightarrow p_{\mathcal{H}}(h(p) = h(q)) \geq p_1$$
$$\|p - q\| \geq R \Rightarrow p_{\mathcal{H}}(h(p) = h(q)) \leq p_2$$

понятно, что хотим $p_1 > p_2$

LSH алгоритм

Построение:

1. Выберем L функций семейства $\mathcal{H}^k$
2. Поделим все данные на кусочки с одинаковыми значениями $g_i = (h_{i,1}, \ldots, h_{i,k})$

Поиск по точке q:

1. Выкинем $j \sim U\{1, \ldots, L\}$
2. Найдем значение $\}_j(q)$ и соответствующий этому значению "кусочек"
3. Линейно поищем в "кусочке"
4. Одно из 2-х:
   1. повторим 1-3 пока не найдется L' точек (включая дубли) ближе чем R.
   2. переберем все L функций

Свойства LSH

Первая стратегия с $L' = 3L$, для любой пары $(R, \delta), \exists L = O(N^{\frac{ln p_1}{ln p_2}})$, гарантирующий $(c, R) - NN$.

Вторая стратегия гарантирует рандомизированного R ближайшего соседа при $\delta = \delta(k, L)$.

Некоторые известные семейства функций

l_1 возьмем $w \gg R$, $s_i \sim U[0, w], i = 1, \ldots, n$,
$h_{s_1,\ldots,s_n} = ([\frac{(x_1 - s_1)}{w}], \ldots, [\frac{(x_n - s_n)}{w}])$

l_s возьмем $w \gg R$, $r_i \sim N(0, 1)$, $h_{w,r,b} = [\frac{r^T x + b}{w}]$

косинусная мера возьмем $r_i \sim N(0, 1)$, $h_r = sign(x, r)$