Уменшение размерности: обзор

И. Куралёнок

СПб, 2017

Зачем бороться с размерностью?

1. Наглядность: сложно смотреть на 100-мерное пространство
2. Количество данных и надежно подбираемых параметров зависимы
3. Экономим стоимость использования:
   - вычислительные ресурсы на этапе обучения и/или использования;
   - стоимость отдельного измерения.
4. Убрать шум

Проклятье размерности

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

Постановка задачи обучения

С построением фичей: повесим на клиента датчики
Наша цель повесить датчики правильно, зная какую информацию мы хотим получить.
Похоже на glass box

Без построения фичей: льется поток неведомых данных
Хотим выделить сигналы, имеющие отношение к искомому
Похоже на black box

Почему можно уменьшить размерность?

В конструктивной постановке

Датчики могут:

- быть разные, а информация одна и та же;
- быть неудачно поставленны оператором и работать не так, как задумано.

Почему можно уменьшить размерность?

В случае сырых данных

В случае сырых данных, мы по сути должны найти датчики, изучив их свойства по имеющимся данным. Поэтому все предыдущее верно и тут, а кроме этого:

- надо понять где причина, а где следствие;
- отделить свойства передачи данных от информации.

Как можно это делать

Feature selection (whitening)

Пример: проверка стат гипотез, что фича нужна

Feature extraction/construction (kostyl production)

Пример: метод главных компонент (PCA)

Embedded models (velosiped prouction)

Пример: LASSO

Как можно это делать

Feature selection (whitening)

Пример: проверка стат гипотез, что фича нужна
Девиз: меньше фишек — легче думать.

Feature extraction/construction (kostyl production)

Пример: метод главных компонент (PCA)

Embedded models (velosiped prouction)

Пример: LASSO

Как можно это делать

Feature selection (whitening)

Пример: проверка стат гипотез, что фича нужна
Девиз: меньше фишек — легче думать.

Feature extraction/construction (kostyl production)

Пример: метод главных компонент (PCA)
Девиз: Мурзик, ну еще капельку!

Embedded models (velosiped prouction)

Пример: LASSO

Как можно это делать

Feature selection (whitening)

Пример: проверка стат гипотез, что фича нужна
Девиз: меньше фишек — легче думать.

Feature extraction/construction (kostyl production)

Пример: метод главных компонент (PCA)
Девиз: Мурзик, ну еще капельку!

Embedded models (velosiped prouction)

Пример: LASSO
Девиз: само сдохнет.

Область применения

Feature selection

Зачем: оценка новых параметров, наглядность, экономия ресурсов, надежный подбор, прибрать шум.

Когда:

- шум/сигнал (dB) у некоторых совсем поганый
- оценка источника сигнала

Область применения

Feature extraction

Зачем: наглядность (быть аккуратными!), экономия ресурсов обучения и использования (предобработка), прибрать шум, гибкость в построении данных.

Когда:

- несколько факторов про одно, но все глядят в разные стороны из-за особенностей реализации;
- хотим обобщающие факторы;
- понимаем структуру данных;
- хочется потратить время.

Область применения

Embedded models

Зачем: экономия ресурсов использования, убрать шум

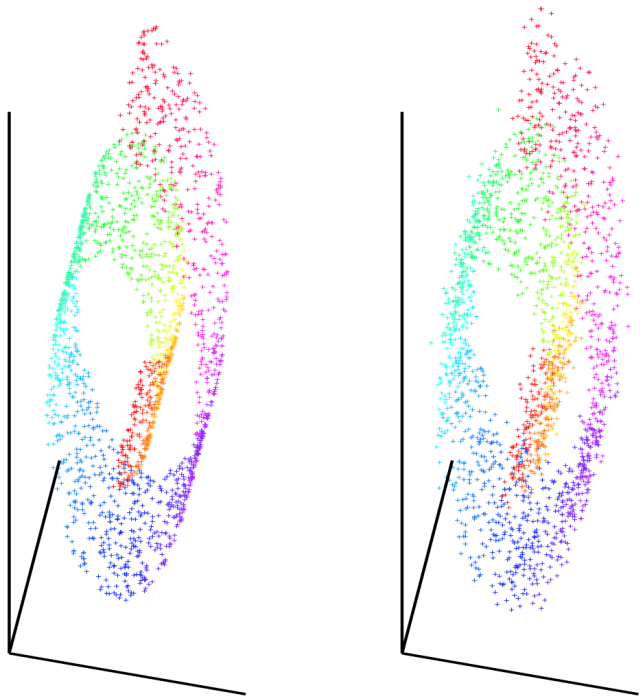
Когда:

- ничего не знаем про данные, зато понимаем как может быть устроено хорошее решение;
- не хотим думать про механизм приемки фичей.

Эффективная размерность

Постановка задачи

Бывают такие ситуации, когда размерность пространства задачи может быть явно меньше чем количество факторов.



Эффективная размерность

"Определение"

Definition (Эффективная размерность)

Будем называть размерность $n' < n$ эффективной размерностью задачи, если существует такое преобразование пространства задачи в $\mathbb{R}^{n'}$, что сохраняет интересные нам свойства

Jonson-Lindestrauss lemma I

Theorem

Для заданного $0 < \epsilon < 1$, *множества точек* $X \supset \mathbb{R}^{n \times m}$, *числа* $k > \frac{\log m}{\epsilon^2}$, *существует линейное преобразование* $f : \mathbb{R}^n \to \mathbb{R}^k$ *такое, что:*

$$(1 - \epsilon)\|u - v\|_2^2 \leq \|f(u) - f(v)\|_2^2 \leq (1 + \epsilon)\|u - v\|_2^2$$

для любой пары $(u, v) \in X^2$.

Jonson-Lindestrauss lemma II

К сожалению лемма довольно tight:

Theorem

Существует множество точек X, для которого потребуется $k = O(-\frac{\log m}{\epsilon^2 \log \epsilon})$.

Но мы помним:

- ограничения только на линейные преобразования;
- есть предположение о полной информации, содержащейся в X.

Как узнать эффективную размерность?

- Спектральная энтропия erank
- Сонаправленность соседей
- Измерение концентрации примеров вблизи точки

Идея: отобразить многомерное пространство в граф, зафиксированной структуры так, чтобы как можно лучше сохранить взаимные расстояния.

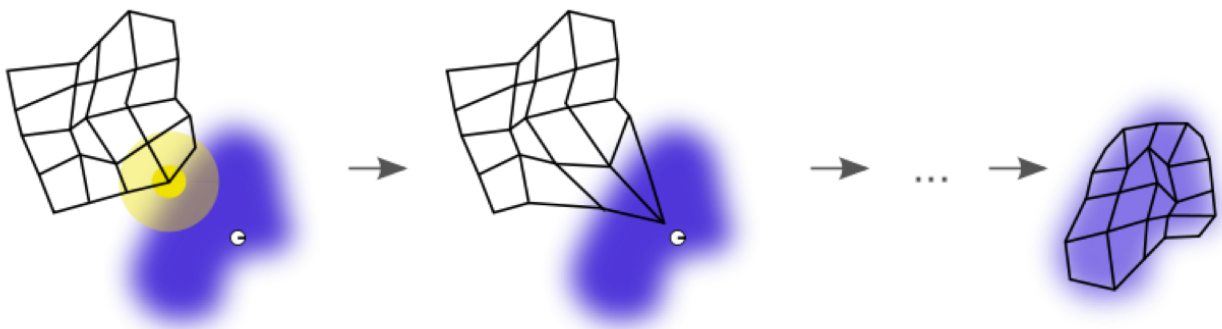
Графы бывают разные:

- Простая 2/3-х мерная сеть
- Шестигранная сетка
- Односвязная “нить”.

SOM

процесс обучения

Можно представить так: кидаем платок и расправляем его.



SOM

алгоритм

Дано: функция расстояния по графу (G), множество точек $(X \in \mathbb{R}^n)$, дискаунт за расстояние (q), дискаунт за итерацию (s)

1. Инициализируем точки, соответствующие узлам графа g_i в $\mathbb{R}^n$
2. Выберем случайную точку x^t в X
3. Находим ближайший g_{i_0}
4. "Двигаем" g_i в сторону точки x^t, по принципу "чем дальше, тем меньше":

$$g_i^{t+1} = g_i^t + q(G(i, i_0), t)s(t)(x_t - g_t)$$

5. Переходим в п.2 пока не сошлось.

SOM

свойства

Что можно с их помощью делать:

- Смотреть на данные “глазами”
- Если из области следует какая-то структура, то можно подобрать ее распределение таким образом
- Аля PCA по 1D кривулинам, полученным с помощью SOM

Проблемы:

- Результат существенно зависит от подбора начальных $g_i \Rightarrow$ зачастую не воспроизводим
- Никаких гарантий сходимости при $s(t) = 1$
- Никаких хороших математических свойств.

Но, it just works!

SOM

рекомендации SOMоводов

- Выбирать начальные значения не по рандому, а по PCA2
- Выбросить выбросы/сгладить движение от дистанции
- Бегать не в одном и том же порядке по множеству
- По минимуму использовать s.

Feature selection

Какие тут есть задачи?

- Понять годится ли нам фича, или можно без нее:
  - как это делать надежно;
  - можно ли сделать подешевле.
- Выбрать оптимальное подмножество факторов.
- Научиться строить модели, предпочитающие поменьше факторов.

Как можно понять нужна или нет фича?

Можно проверить такую гипотезу:

Theorem

Для множеств $X_B \in \mathbb{R}^{n \times m}$ *и* $X_T = (X_B, x) \in \mathbb{R}^{n \times (m+1)}$*, парных последовательностей разбиений этих множеств:*

$$\{(L_{Bt}, V_{Bt})\}_1^\infty, L_{Bt} \cup V_{Bt} = X_B, \forall t$$
$$\{(L_{Tt}, V_{Tt})\}_1^\infty, L_{Tt} \cup V_{Tt} = X_T, \forall t$$

, целевой функции $T(\beta, X)$ *событие*

$$T(\arg\max_\beta T(L_{Bt}), V_{Bt}) \leq T(\arg\max_\beta T(L_{Tt}), V_{Tt})$$

выполняется с вероятностью не менее $1 - \alpha$*.*

Немного о свойствах проверки

У подобной постановки есть несколько свойств:

- сразу вставили повторную выборку, чтобы не париться;
- заметим, что подбираем прямо β, что означает зависимость метода от способа оптимизации;
- конкретный способ проверки подобной гипотезы зависит от распределения $T(\arg\max_\beta T(L_{Bt}), V_{Bt})$;
- считаем, что проверяемый фактор не влияет на это распределение;
- есть надежда на то, что исходный X несмещен относительно генеральной совокупности по $T(\arg\max_\beta T(L_{Bt}), V_{Bt})$
- исходим из независимости $T(\arg\max_\beta T(L_{Bt}), V_{Bt})$ по t, что, конечно, не так :).

Практическая реализация проверки

Что нам нужно определить чтобы такое реализовать?

- Понять как делить X.
- Выяснить какой критерий использовать.
- Найти необходимое количество разбиений для достижения уровня значимости α.
- Нащупать границы применимости.

Как делить X

Мы делили апельсин...

Можно через BS, но мы побаиваемся зависимости, и делаем CV. В CV обычно встает вопрос соотношения мощности L и V. Есть две альтернативы:

- Сделать L поменьше
- Сделать T поменьше

$$T(\arg\max_{\beta} T(L_{Bt}), V_{Bt}) \leq T(\arg\max_{\beta} T(L_{Tt}), V_{Tt})$$

Как делить X

Мы делили апельсин...

Можно через BS, но мы побаиваемся зависимости, и делаем CV. В CV обычно встает вопрос соотношения мощности L и V. Есть две альтернативы:

- Сделать L поменьше
  Увеличит variance β и, соответственно, может повлиять на дисперсию случайных величин справа и слева, при фиксированном.
- Сделать T поменьше

$$T(\arg\max_{\beta} T(L_{Bt}), V_{Bt}) \leq T(\arg\max_{\beta} T(L_{Tt}), V_{Tt})$$

Как делить X

Мы делили апельсин...

Можно через BS, но мы побаиваемся зависимости, и делаем CV. В CV обычно встает вопрос соотношения мощности L и V. Есть две альтернативы:

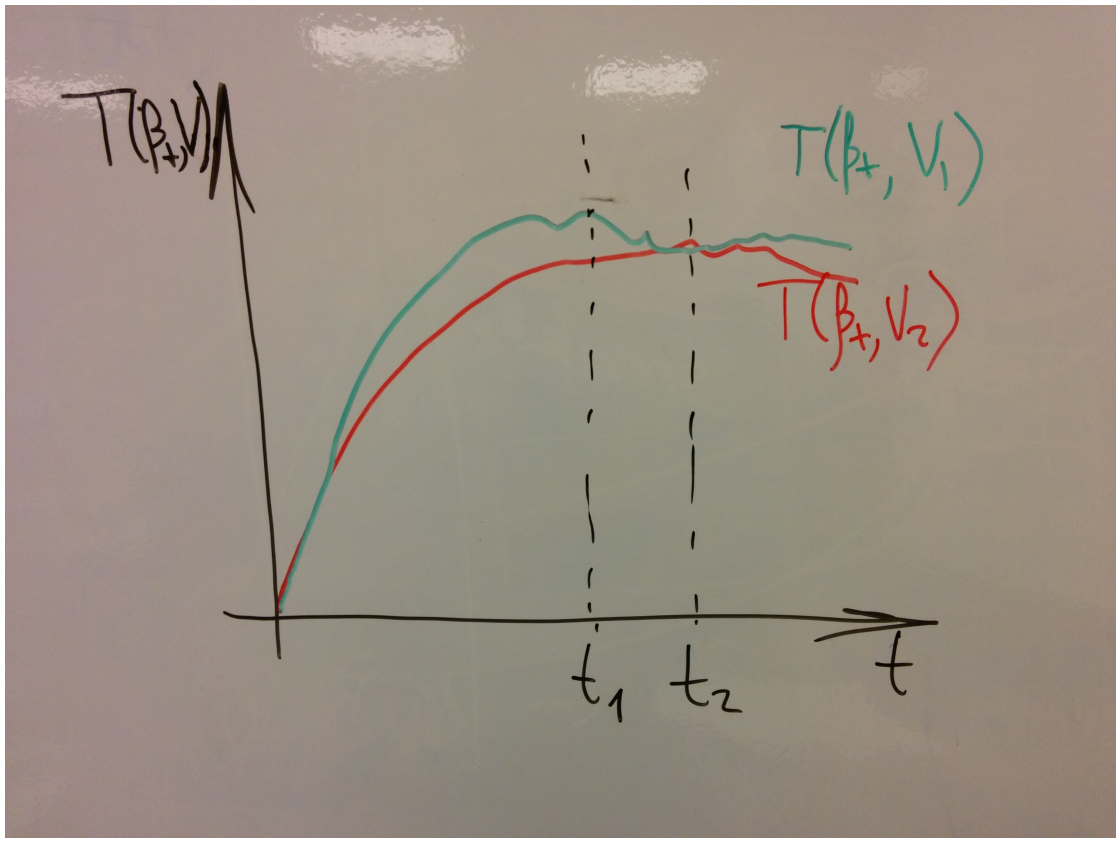
- Сделать L поменьше
  Увеличит variance β и, соответственно, может повлиять на дисперсию случайных величин справа и слева, при фиксированном.
- Сделать T поменьше
  Обычно мы строим T используя информацию о независимости наблюдаемых точек, что приводит к обратной зависимости дисперсии от количества точек.

$$T(\arg\max_{\beta} T(L_{Bt}), V_{Bt}) \leq T(\arg\max_{\beta} T(L_{Tt}), V_{Tt})$$

$\Rightarrow$ надо найти баланс при котором достигается $\min var(T(\arg\max_{\beta} T(L_{Bt}), V_{Bt}))$

Какой критерий выбрать?

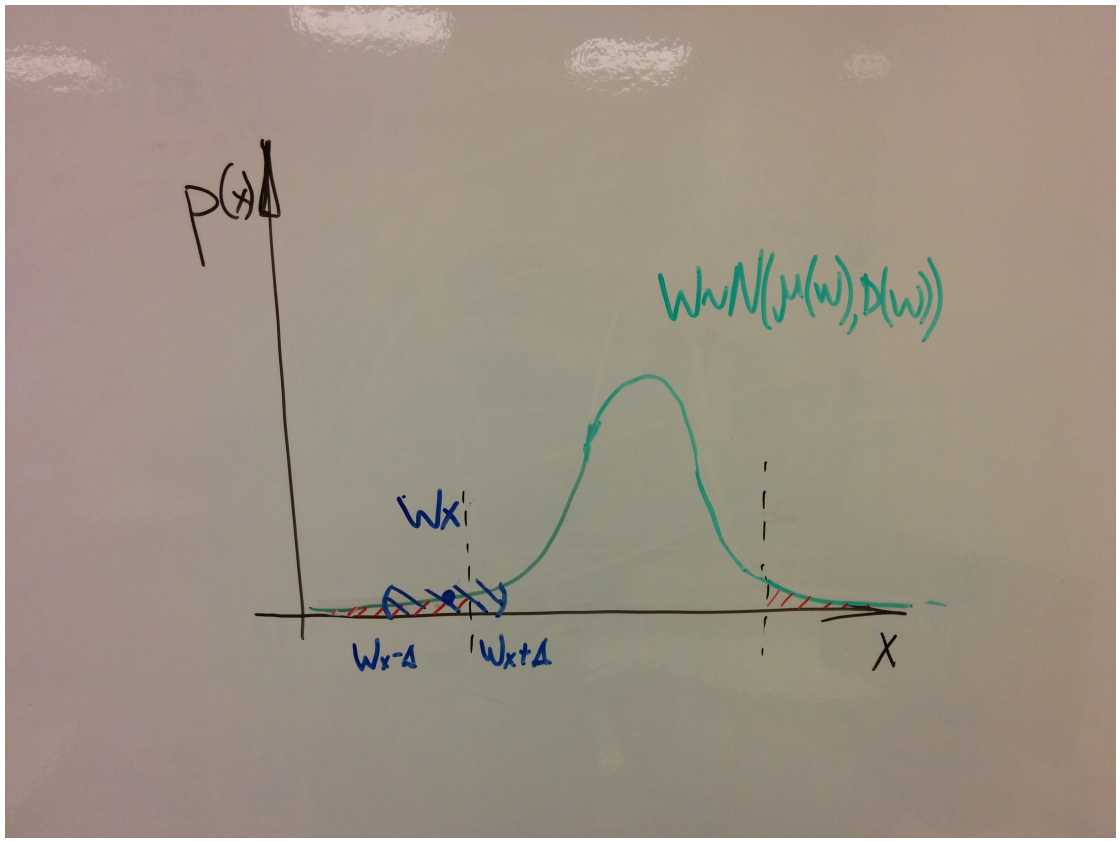
Зависит от метода:



Можно не париться и взять *WX*-тест, который работает почти всегда.

Сколько разбиений достаточно?

Само по себе значение статистики является случайной величиной со своей дисперсией, это надо учитывать.



Зависит дисперсия от количества разбиений. Сделаете 10 — будет гигантским, сделаете 1000 — будет точнее. У нас 200.

Когда оно все одно лажает?

Тем не менее все это добро может и не работать.

1. Значения W_t зависимы
2. Все еще остается α с которой эта штука будет принимать одинаковые DS за ухудшения
3. Мы могли неверно подобрать какой-нить магический параметр

$\Rightarrow$ все равно надо тестировать

Wrapper vs. Filter

Если использовать исходную оптимизацию, можно поседеть. На чем можно сэкономить?

- Использовать менее точную оптимизацию.
- Поменять целевую функцию в сторону более простую (e.g. парную на точечную)
- Использовать другую модель обучения, похожую на боевую
- Использовать тот факт, что мы проводим много последовательных разбиений одного и того же множества
- etc.

Все подобные телодвижения переводят нас из класса wrapper методов (честных), в класс filter (нечестных).

Как можно тестировать приемку факторов?

- Изменения не несущие информацию:
  - добавить рандомные фичи;
  - добавить дубли;
  - пошуметь на дублях;
  - использовать неинформативные преобразования.
- Поведение на исходном сигнале с разным уровнем шума
- Известные хорошие фичи

Немного статистики из жизни Яндекс

За первый год использования автоматизированной проверки факторов было проверено 50000 гипотез.
На проверку факторов тратится $\sim$ 10000000 машиночасов[1]
Вся эта работа позволяет поставить работу над факторами в конвейерный режим и получать стабильный прирост качества.

Подробнее можно почитать на хабре по тегу *FML*.

[1] сейчас больше, но не хочу показывать

Выбираем оптимальное множество факторов

К сожалению, задача *NP*-полная, так что нас ничего не спасет из-за NFL. Можно попробовать сделать приблизительно:

- Greedy forward selection
- Greedy backward elimination

Получается не очень, если делать без знания о методе оптимизации.

Пара слов про embedded

Мы уже знаем пару способов как делать embedded.
Вот пара способов:

- Akaike information criterion (и его исправленная, неработающая версия)
- Bayesian information criterion
- Minimum description length
- l_q, для $q < 2$.

Что мы сегодня узнали

- Есть такая задача — фичи убирать
- К ней можно подступаться с разными девизами
- Рассмотрели задачу выбора факторов:
  - узнали как проверить одиночное изменение (а точно одиночное?);
  - поняли как все там на ладан дышит;
  - посетовали на то, что оптимальный набор фичей сделать сложно;
  - вспомнили немного принципов по которым можно построить embedded model для feature selection.