

Subject: Stanford CS229 Machine Learning, Lecture 7, Kernels

Date: from October 5, 2024 to October 8, 2024

Contents

Stanford CS229 Machine Learning, Kernels, 2022, Lecture 7

Link on YouTube: Stanford CS229 Machine Learning, Kernels, 2022, Lecture 7

Feature Function

Introduction: Cubic Polynomial Regression, where data set is $\{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$

在基本线性回归(Linear Regression)模型 $h_{\theta}(x) = \theta x + \theta_0$ 中^a, 我们仅仅使用了 x 自身进行数据拟合, 但我们常需要更复杂的模型, 例如三次多项式(cubic polynomial)模型:

$$h_{\theta}(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x + \theta_0, x \in \mathbb{R}$$

显然模型 $h_{\theta}(x)$ 关于 x 是非线性的, 但关于参数 θ 却是线性的^b。我们只需做出一些改变, 就可以将关于 x 的非线性模型变为关于 θ 的线性模型。

^a其中 θ 代表全体参数

^b我们关注的核心实际上是在参数空间中对 θ 进行优化

Feature Map / Feature Extractor (cubic polynomial regression case)

定义函数 $\phi: \mathbb{R} \rightarrow \mathbb{R}^4$, $\phi(x) = [1 \ x \ x^2 \ x^3]^T$, 这样模型就可以写为

$$h_{\theta}(x) = [\theta_0 \ \theta_1 \ \theta_2 \ \theta_3] [1 \ x \ x^2 \ x^3]^T = \theta^T \phi(x)$$

如此一来我们原本的模型输入从 $[1 \ x]$ (二维)变成 $\phi(x)$ (四维), 因此我们构建了新数据集:

$$\{(\phi(x^{(1)}), y^{(1)}), \dots, (\phi(x^{(n)}), y^{(n)})\}$$

其中函数 $\phi(\cdot)$ 被称为特征提取函数(feature function / feature extractor), $\phi(x)$ 被称为特征(features), x 被称为属性(attribute)。如果 $x \in \mathbb{R}^d$, 那么相应的 $\theta \in \mathbb{R}^d$; 在经过 $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^p$ 的作用后, 相应的 $\theta \in \mathbb{R}^p$ 。

Kernel Trick / Kernelized

Basic Settings

在新数据集上做线性回归, 并使用梯度下降(GD)进行优化, 学习率为 α , 此时损失函数和参数 θ 的更新过程分别为式(1a)(1b)

$$loss = \frac{1}{2} \sum_{i=1}^n \left(y^{(i)} - \theta^T \phi(x^{(i)}) \right)^2 \quad (1a)$$

$$\theta := \theta + \alpha \cdot \sum_{i=1}^n (y^{(i)} - \theta^T \phi(x^{(i)})) \phi(x^{(i)}) \quad (1b)$$

考虑 $x \in \mathbb{R}^d$, 使用三次多项式回归, 那么 $\phi(x) \in \mathbb{R}^p$, 其中 $p = 1 + |\{x_i \cdot x_j\}| + |\{x_i \cdot x_j \cdot x_k\}| = 1 + d + d^2 + d^3$ (考虑重复情形)。因此每一次参数 θ 的更新都会有 $O(np)$ 的计算量, 这是十分巨大的。

Kernel Trick

Proposition 1 (Key Observation). 若 $\theta^0 = 0$, 那么 θ 可以表示为 *features* 的线性组合, 即

$$\theta = \sum_{i=1}^n \beta_i \phi(x^{(i)}) \quad (2)$$

其中 $\beta_1, \dots, \beta_n \in \mathbb{R}$, θ^0 是 θ 的初始值。

Proof. 我们使用数学归纳法(induction)。

当 $\text{iteration} = 0$ 时, $\theta = 0 = \sum_{i=1}^n 0 \cdot \phi(x^{(i)})$ 显然是 *features* 的线性组合;

当 $\text{iteration} = 1$ 时, $\theta := \theta + \alpha \sum_{i=1}^n (y^{(i)} - \theta^T \phi(x^{(i)})) \phi(x^{(i)}) = \alpha \sum_{i=1}^n y^{(i)} \phi(x^{(i)})$, 此时 $\beta_i := \alpha y^{(i)}$

假设 $\text{iteration} = t$ 时结论成立, 即 $\theta = \sum_{i=1}^n \beta_i \phi(x^{(i)})$, 则 $\text{iteration} = t + 1$ 时

$$\begin{aligned} \theta &:= \theta + \alpha \sum_{i=1}^n (y^{(i)} - \theta^T \phi(x^{(i)})) \phi(x^{(i)}) = \sum_{i=1}^n \left(\left(\beta_i + \alpha (y^{(i)} - \theta^T \phi(x^{(i)})) \right) \phi(x^{(i)}) \right) \\ \beta_i &:= \beta_i + \alpha (y^{(i)} - \theta^T \phi(x^{(i)})) \end{aligned} \quad (3)$$

□

Trick 1: parameters storage 已知 $\theta \in \mathbb{R}^p$, 但是现在我们不需要直接存储 θ 而是存储系数 β_i , 这只需要 n 个存储, 不妨假设 $p \gg n$, 那么这将带来一些计算量的降低。

Problem 1: still related to θ 从式(3)可以看到 β 的更新实际上与 θ 相关, 仍需 $O(p)$ 的计算量, 因此我们需要找到 β 只依赖于自身的更新模式:

$$\begin{aligned} \beta_i &:= \beta_i + \alpha (y^{(i)} - \theta^T \phi(x^{(i)})) = \beta_i + \alpha \left(y^{(i)} - \left(\sum_{j=1}^n \beta_j \phi(x^{(j)}) \right)^T \phi(x^{(i)}) \right) \\ &= \beta_i + \alpha (y^{(i)} - \sum_{j=1}^n \beta_j \langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle) \end{aligned} \quad (4)$$

Problem 2: inner product costs a lot 但是可以看到在计算内积 $\langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle$ 时, 仍然需要 $O(np)$ 的计算量, 下面有两个简化方式(**Trick 2**):

① **preprocessed:** $\langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle$ 可以被预处理(preprocessed)并存储, 因为 $\forall i, j$, 内积计算过一次后就无需重复计算了

② **formal compute:** 无需精确地将复杂的 $\phi(\cdot)$ 带入 $\langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle$ 中也可以进行计算:

$$\begin{aligned} \langle \phi(x), \phi(z) \rangle &= [1, x_1, \dots, x_d, x_1^2, \dots, x_d^2, x_1^3, \dots, x_d^3] [1, x_1, \dots, x_d, x_1^2, \dots, x_d^2, x_1^3, \dots, x_d^3]^T \\ &= 1 + \sum_{i=1}^d x_i z_i + \sum_{i=1, j=1}^d x_i x_j \cdot z_i z_j + \sum_{i, j, k=1}^d x_i x_j x_k z_i z_j z_k \\ &= 1 + \langle x, z \rangle + \sum_{i=1}^d x_i z_i \sum_{j=1}^d x_j z_j + \sum_{i=1}^d x_i z_i \cdot \sum_{j=1}^d x_j z_j \cdot \sum_{k=1}^d x_k z_k = 1 + \langle x, z \rangle + \langle x, z \rangle^2 + \langle x, z \rangle^3 \end{aligned}$$

Algorithm and Time complexity

可以看到计算单个内积的时间复杂度为 $O(d)$ 。称 $K: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$, $K(x, z) = \langle \phi(x), \phi(z) \rangle$ 为核函数(kernel function)。这样的算法称为Kernel method for regression, 最终的算法为:

Algorithm 1 Algorithm for Kernel Computation and β Update

```
1: Input: Data points  $\{x^{(i)}\}_{i=1}^n$ , labels  $\{y^{(i)}\}_{i=1}^n$ , learning rate  $\alpha$ 
2: Initialize:  $\beta = 0 \in \mathbb{R}^n$ 
3: for  $i = 1, \dots, n$  do
4:   for  $j = 1, \dots, n$  do
5:     Compute the kernel  $K(x^{(i)}, x^{(j)}) = \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$ 
6:   end for
7: end for
8: for  $i = 1, \dots, n$  do
9:    $\beta_i \leftarrow \beta_i + \alpha \left( y^{(i)} - \sum_{j=1}^n \beta_j K(x^{(i)}, x^{(j)}) \right)$ 
10: end for
```

考虑维度 i, j 后, 计算所有的kernel的总时间复杂度为 $O(n^2 d) = O(n^2)$, 因此对于 β , 每一次迭代的时间复杂度为 $O(n^2)$ 。可以看到单次迭代 β 的时间复杂度变化为 $O(np) \rightarrow O(n^2)$, 因此若 $n \ll p$, 那么就会带来很大的计算量提升。^a

^a整个算法的时间复杂度为 $O(n^2 d) + O(n^2 \cdot T)$, 其中 T 是迭代次数

Design Kernel Function

Change the algorithm flow

从上述推导过程来看, 我们将 $K(\cdot, \cdot)$ 从特征函数的内积形式化成为原数据的内积进而减少了计算量。在进行测试时, 给定一个 x , 根据式(3)和Proposition 1需要计算

$$\theta^T \phi(x) = \left(\sum_{i=1}^n \beta_i \phi(x^{(i)}) \right)^T \phi(x) = \sum_{i=1}^n \beta_i K(x^{(i)}, x)$$

^a, 因此我们的算法最终仅与核函数 K 有关, 而核函数是特征函数的内积形式, 因此显示定义了特征函数就显式定义了核函数, 反过来直接定义核函数也就隐式定义了特征函数, 因此^b:

Design a good $\phi \Leftrightarrow$ Design a good K

Theorem 1 (necessary and equivalent condition for designing a K). Suppose $x^{(1)}, \dots, x^{(n)}$ are n data points, and let $K \in \mathbb{R}^{n \times n}$ be kernel matrix, in which $K_{ij} = K(x^{(i)}, x^{(j)})$. K is a valid kernel function $\Leftrightarrow \forall x_1^{(n)}, \dots, x^{(n)}$, the kernel function K is PSD (positive semi-definite).

Proof. 暂略 □

如此一来我们的工作流程就反过来变为了:

设计一个好的kernel function $K \rightarrow$ 确定 K 有效(通过Thm 1 或者解出 $\phi(\cdot)$) \rightarrow 运行算法

^a可以看到此时测试时仍然需要训练数据, 与之前训练好之后就无需训练数据只需训练好的参数的情况不同

^b这里的 K 的设计要make sense, 要有意义

Extended content

Extended content

常用的核函数有

1. $K(x, z) = (\langle x, z \rangle + c)^k$, for $k = 2, \phi(x) = \begin{bmatrix} c \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \vdots \\ x_1^2 \\ \vdots \\ x_d^2 \end{bmatrix}$
2. Gaussian Kernel: $K(x, z) = \exp(-\frac{\|x-z\|_2^2}{2\sigma^2}) = \langle \phi(x), \phi(z) \rangle$, 但是实际上 $\phi(\cdot)$ 很复杂, 是一个无穷维形式

有时特征也被视为相似性度量(similarity metric), 可以将 $K(x, z)$ 看作一种测量 x 和 z 的方式(measure)

kernel method 真正的改变是将 $O(np) \rightarrow O(n^2)$, 当 $n \ll p$ 时这较有效, 但是现在的数据集往往很大, 因此kernel method 现在并不是一个很高效和常用的方法; 同时核函数的设计并不很容易, 同时可解释性也是一个困难。事实上NN也可以写为 $\theta^T \phi_w(x)$, 其中 w 代表NN中的参数, 而此 ϕ_w 是直接从数据中学习得到的, 而非人工设计的, 但是有效性却出奇的好。

First updated: October 8, 2024

Last updated: December 23, 2024