CS229 Machine Learning, Supervised learning setup, 2022, Lecture 2

YouTube: Stanford CS229 Machine Learning, Supervised learning setup, 2022, Lecture 2

Definitions

Supervised Learning

监督学习的一个重要特征是数据集中含有标签数据。以预测任务(prediction)为例，我们希望找到一个映射$h: x \to y$，将数据$\boldsymbol{x}$尽量准确地对应于y，例如判断一张图片($\boldsymbol{x}$)中是否是猫(y)，一段文本($\boldsymbol{x}$)是否含有仇恨语言(y)，通过一系列特征($\boldsymbol{x}$)预测房价(y)等。

监督学习的数学表达如下：

given: 给定训练数据集(training set)

$$D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \cdots, (\boldsymbol{x}^{(n)}, y^{(n)})\}, \quad \text{其中} \boldsymbol{x}^{(i)} \in X \subseteq \mathbb{R}^m, y^{(i)} \in Y$$

do: 找到一个"好"[a]的映射[b]$h: x \to y$.

Note 1. 我们想要的并不完全是h能够在训练集D上表现好，我们更希望在D之外的数据上h也能够有很好的表现，即关注所谓的泛化能力(***generalization***)。并且这里有一个隐含的前提假设是D满足的分布与真实世界的分布相同，详见附录*A*.

如果我们的标签y是离散(discrete)的，那么就称该监督学习任务是分类任务(**classification**)，例如判断一张图片中的东西是否是猫咪，输出0表示不是，输出1表示是.如果y是连续的(continuous)，那么称之为回归任务(**regression**)，例如著名的波士顿房价预测[c].

[a] 如何称为"好"以及怎么构建"好"的映射是一个重要的问题，将在后续课程讲解

[b] 这里使用字母h的原因是也可以称之为一个假设(hypothesis)

[c] Kaggle 数据集, Kagggle 上的实践代码

Linear Regression

Linear Regression Model

线性回归是最简单的回归模型，其是指将h表示为所有选定特征的线性加和，即：

$$h(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_m x_m = \sum_{i=0}^{m} \theta_i x_i \tag{1}$$

其中，数据特征向量$\boldsymbol{x} = (x_1, x_2, \cdots, x_m)$，通常为了方便我们引入一个偏置项$x_0 = 1$，从而可以将模型简洁地表示为$h(\boldsymbol{x}) = \sum_{i=0}^{m} \theta_i x_i$.(注意此时$\boldsymbol{x}$已经增加一维)

在模型训练时，我们所做的是将h作用于各数据$\boldsymbol{x}^{(i)}$，将之对应于相应的标签$y^{(i)}$，如果将所有的数据集统一表达，就是下面的矩阵形式：

$$h(X) = \theta X \approx Y, \quad X = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(m)} \end{bmatrix}, Y^T = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}, \theta^T = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

一个二维的简单情形如图1所示，其中线性模型为$h(x) = \theta_0 + \theta_1 x$，$\theta_0$反映了该直线的截距，$\theta_1$反映了斜率。

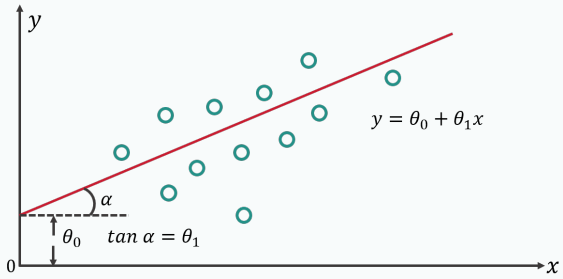

Figure 1: supervised learning in 2 dimansion case

Loss Function

现在我们已有具体模型的抽象数学表达式(1)，但参数$\boldsymbol{\theta} = (\theta_0, \cdots, \theta_n)$现在都是未知的。为获得这些参数可以从优化的视角去看：如果我们有了一种量化一组参数$\{\theta_i\}_{i=0}^n$好坏的方式(或称度量)$L(\boldsymbol{\theta})$，并且$L(\boldsymbol{\theta})$越小代表训练集上$h_{\boldsymbol{\theta}}(\boldsymbol{x})$与$y$差距越小，那么在众多的参数选择中，我们需要找到一组最好(或者一定程度上较好)的参数$\{\theta_i^*\}_{i=0}^n$使得$L(\boldsymbol{\theta}^*)$最小(或较小)。因此我们现在需要一种度量参数好坏的方式，这就是损失函数(**loss function**)

最简单的损失函数是均方误差(**Mean Squared Error, MSE**)：

$$L(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 \tag{2}$$

其中$\frac{1}{2}$的作用是在对$L(\boldsymbol{\theta})$后与2约去，更简洁。并且从MSE的定义也可以看出，$L(\boldsymbol{\theta})$越小代表在总体平均的意义下，在训练集上$L(\boldsymbol{\theta})$能够使得$h_{\boldsymbol{\theta}}(\boldsymbol{x})$越接近于$y$.

Gradient Decent

我们知道一个函数的负梯度方向是使其函数值下降的方向，因此我们在寻找$\boldsymbol{\theta}^* = \arg\min L(\boldsymbol{\theta})$时可以首先随机选择一个$\boldsymbol{\theta}^{(0)}$，然后沿着$L(\boldsymbol{\theta}^{(0)})$的负梯度方向"走一步"得到$\boldsymbol{\theta}^{(1)}$，依此类推慢慢向$\boldsymbol{\theta}^*$靠近，示意图如图2所示，其中黄色为$L(\boldsymbol{\theta}^{(0)})$的负梯度方向.

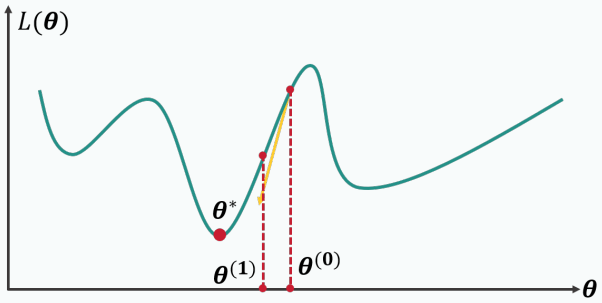

Figure 2: gradient decent

用数学形式写下来就是：

$$\begin{aligned} \boldsymbol{\theta}^{(0)} &= \boldsymbol{\theta}^{(0)} \\ \theta_j^{(t+1)} &= \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} L(\theta_j^{(t)}),\ j = 1, \cdots, m \end{aligned} \tag{3}$$

其中α就是上文所说的“走”一步的“步长”.

上面的算法被称为梯度下降法(**Gradient Decent, GD**)，实际上GD会遇到几个问题：

1. 初始值$\boldsymbol{\theta}^{(0)}$选取的不够好很可能无法得到全局极小点，而陷入局部极小，即鞍点(saddle point),如图3区域II所示，使用GD只能靠近鞍点$\tilde{\boldsymbol{\theta}}$。一种解决方案是引入随机性，使得下一步的方向有可能能够逃出鞍点.
2. 由于步长α是固定的，因此有可能出现区域I中“反复横跳”的情形，造成算法的收敛非常困难。一种解决的方案是将步长α随着迭代步数t的增加不断减小.

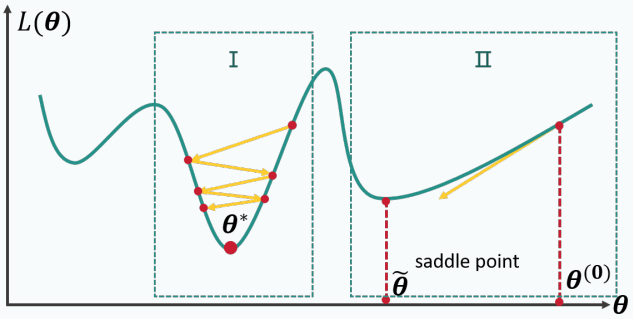

Figure 3: GD problems

对于式(3)，我们完整写出更新过程就是：

$$\begin{aligned} \theta_j^{(t+1)} &= \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} L(\boldsymbol{\theta}^{(t)}) \\ &= \theta_j^{(t)} - \alpha \sum_{i=1}^{n} \frac{1}{2} \frac{\partial}{\partial \theta_j} \left(h_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{x}^{(i)}) - y^{(i)} \right)^2 \\ &= \theta_j^{(t)} - \alpha \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{x}^{(i)}) - y^{(i)} \right) \frac{\partial}{\partial \theta_j} h_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{x}^{(i)}) \\ &= \theta_j^{(t)} - \alpha \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{x}^{(i)}) - y^{(i)} \right) x_j^{(i)},\ j = 1, \cdots, m \end{aligned} \tag{4}$$

其中最后一步利用了$h(\boldsymbol{x})$的定义式(1).

Full Batch v.s. Stochastic Mini Batch

式(4)的一个问题是每一步的更新都需要对训练集D上的n个数据全部计算一次，这一计算量有时是非常大的，也就是说梯度下降每更新一步都耗时较长。因此下面介绍小批次的梯度下降算法.

Batch: 在机器学习中，“batch”指的是在一次迭代中用于训练模型的一组样本。一次训练使用全部训练数据就是使用Full Batch，如果将一份训练集随机分为若干等分，

每次使用一份进行训练，就是使用Mini Batch。其重要优势是可以在GPU上实现并行化(parallelism)。下面是其数学表示。

我们将训练集D均分为d份，分别记为$D_1, D_2, \cdots, D_d$，其中$D_i \cap D_j = \varnothing (i \neq j)$，$\bigcup_{k=1}^{d} D_k = D$。这样在每一个mini batch D_k上，我们都有：

$$\frac{\partial}{\partial \theta_j} L_k(\boldsymbol{\theta}^{(t)}) = \sum_{i \in D_k} \left(h_{\boldsymbol{\theta}^{(t)}}(\boldsymbol{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

显然对于式(4)中的$\frac{\partial}{\partial \theta_j} L(\boldsymbol{\theta}^{(t)})$有：

$$\frac{\partial}{\partial \theta_j} L(\boldsymbol{\theta}^{(t)}) = \sum_{k=1}^{d} \frac{\partial}{\partial \theta_j} L_k(\boldsymbol{\theta}^{(t)})$$

由于各个mini batch 中的梯度计算是可以同时进行的(即并行)，因此较full batch 更高效。

A Appendix

Appendix

泛化能力的假设前提实际上是训练集中数据的分布与现实中的分布是相同的，例如训练集是几千张猫咪的照片，每张照片由1024×1024个像素点构成，所有像素点的数值会满足一个分布，我们希望这个分布与全世界所有的猫咪的照片的分布是一样的。但是，实际上这永远不可能达到，这个分布也许只有上帝才知道，因此这个假设永远是错误的，但是对于这样的一个模型，如果效果足够好当然是可以接受的，这也就是George Box 1976年所说的名言[1]：

All models are wrong, but some are useful.

但是这个前提假设为我们提供了一些显然的指导，例如如果训练集全部是猫咪的照片，那么用在此训练集上训练好的模型去用于小狗的照片，那么结果可想而知会很糟糕。

Last updated: October 31, 2024

References

[1] George EP Box. Science and statistics. *Journal of the American Statistical Association*, 71(356):791–799, 1976.