CS229 Machine Learning Note 1

Supervised Learning: Linear Regression

Basic Terminologies

- $x^{(i)}$: **Input features**
- $y^{(i)}$: **Output/Target variable**
- A pair $(x^{(i)}, y^{(i)})$ is called a training example
- A list of n training examples $\{(x^{(i)}, y^{(i)}); i = 1, \cdots, n\}$ is called a **training set**
- h: **Hypothesis** or **Model**. A function that maps input features to output/target variable
- Training process is like this:

```
        Training set
            ↓
        Training
            ↓
    x -> Hypothesis -> predicted output
```

- We call the learning problem **regression problem** when the output variable y is continuous-valued
- We call the learning problem **classification problem** when the output variable y is discrete-valued

1. Linear Regression

Approximate the hypothesis h with a linear function of x:

$$h_\theta(x) = \theta_0(x_0) + \theta_1 x_1 + \theta_2 x_2 . \cdots$$

where θ_is are the **parameters** (also **weights**) parameterizing the space of linear functions

Introduce the convention $x_0 = 1$ to simplify the notation (as **intercept term**), so that:

$$h_\theta(x) = \theta^T x = \sum_{i=0}^{n} \theta_i x_i$$

where θ, x are $(n+1)$-dimensional vectors, and d is the number of features.

Define the **cost function** (also called **loss function**) as:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$$

where n is the number of training examples.
*This function is called **Ordinary Least Squares**.*

1.1 LMS Algorithm

Choose θ to minimize $J(\theta)$.
Mathematically, it can be done by solving zero points of partial derivatives. However, this is not feasible for computer or in high-dimensional space.
Gradient Descent
Starts with some initial guess $\theta^{(0)}$, and iteratively update θ by:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad \text{for } j = 0, 1, \cdots, n$$

α is the **learning rate** (step size).
In practice, α is chosen by **trial and error**, e.g. 2's exponential.
Work out the RHS:

$$\begin{aligned}\frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}\left(\frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)})-y^{(i)})^2\right)\\ &= \sum_{i=1}^{n}(h_\theta(x^{(i)})-y^{(i)})\frac{\partial}{\partial\theta_j}(h_\theta(x^{(i)})-y^{(i)})\\ &= \sum_{i=1}^{n}(h_\theta(x^{(i)})-y^{(i)})\frac{\partial}{\partial\theta_j}(\theta^T x^{(i)}-y^{(i)})\\ &= \sum_{i=1}^{n}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}\end{aligned}$$

(*Here sum and partial derivatives are interchangeable as they are both linear operations*)

For a single training example $(x^{(i)}, y^{(i)})$, the update rule is:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

This gives the **LMS (Least Mean Squares) update rule** (a.k.a. Widrow-Hoff rule) with several properties:

- The magnitude of the update is proportional to the error

Two ways to modify to multiple training examples:

1. **Batch Gradient Descent**: Update θ using the average of the gradients over all training examples:

$$\begin{aligned}&\text{Repeat until convergence:}\{\\ &\theta_j := \theta_j + \alpha\sum_{i=1}^{n}(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}\\ &\}\end{aligned}$$

Grouping into vector form:

$$\theta := \theta + \alpha\sum_{i=1}^{n}(y^{(i)} - h_\theta(x^{(i)}))x^{(i)}$$

J is a convex function, so batch gradient descent will always converge to the global minimum (for small enough α).
Disadvantages:

- Each step of gradient descent requires a sum over the entire training set, which can be very expensive if the training set is large
- Can be slow to converge

2. **Stochastic Gradient Descent**: Update θ using only one training example at each step:

```
Loop{
    for $i = 1$ to $n${
        $\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$
    }
}
```

Grouping into vector form:

$$\theta := \theta + \alpha(y^{(i)} - h_\theta(x^{(i)}))x^{(i)}$$

Features:

- No need to scan the entire training set to perform each update
- Gets θ close to the minimum (good approximation)
- In practice α is decreased with time (e.g. $\alpha = \frac{const_1}{iteration+const_2}$) to guarantee convergence
- Halt when J has no significant decrease.

For these reasons, particularly when the training set is large, stochastic gradient descent is often preferred over batch gradient descent.

1.2 The normal equations

1.2.1 Matrix Derivatives

For Function $f : \mathbb{R}^{n\times d} \to \mathbb{R}$ mapping from n-by-d matrices to read numbers, we define the **derivative of f** with respect to A as:

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \cdots & \frac{\partial f}{\partial A_{1d}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{n1}} & \cdots & \frac{\partial f}{\partial A_{nd}} \end{bmatrix}$$

$$\implies \nabla_A f_{ij} = \frac{\partial f}{\partial A_{ij}}$$

where A_{ij} denotes the (i, j)-th entry.

1.2.2 Least squares revisited

Rewrite J in matrix vectorial notation:
Given a training set, define the **design matrix** X to be the n-by-d matrix (if consider *intercept term*, n by $d+1$) that contains the training examples' input values in its rows:

$$X = ((x^{(1)})^T (x^{(2)})^T \dots (x^{(n)})^T)^T$$

The target values vector:

$$\vec{y} = (y^{(1)} y^{(2)} \dots y^{(n)})^T$$

$$\implies X\theta - \vec{y} = \begin{bmatrix} (x^{(1)})^T\theta - y^{(1)} \\ \vdots \\ (x^{(n)})^T\theta - y^{(n)} \end{bmatrix} = \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(n)}) - y^{(n)} \end{bmatrix}$$

Since $z^T z = \sum_i z_i^2$:

$$\frac{1}{2}(X\theta - \vec{y})^T (X\theta - \vec{y}) = J(\theta)$$

Find derivatives with respect to θ to minimize J:

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta-\vec{y})^T(X\theta-\vec{y}) \\ &= \frac{1}{2}\nabla_\theta((X\theta)^TX\theta-(X\theta)^T\vec{y}-\vec{y}^TX\theta+\vec{y}^T\vec{y}) \\ &= \frac{1}{2}\nabla_\theta(\theta^T(X^TX)\theta-2(X^T\vec{y})^T\theta) \quad \text{discard items without } \theta \\ &= \frac{1}{2}(2X^TX\theta-2X^T\vec{y}) \\ &= X^TX\theta-X^T\vec{y}\end{aligned}$$

In step 3, $(X\theta)^T\vec{y}=(X^T\vec{y})^T\theta$ *because it is a scalar.*
In step 4, we use property $\nabla_x x^TAx=2Ax$ for **symmetric A** ($A^T=A$)

$$\frac{\partial x^TAx}{\partial x}=(\frac{\partial x}{\partial x})^TAx+x^T\frac{\partial Ax}{\partial x}=Ax+x^TA=Ax+A^Tx=2Ax$$

Thus obtain the **normal equations**

$$X^TX\theta=X^T\vec{y}$$

θ is given in closed form by:

$$\theta=(X^TX)^{-1}X^T\vec{y}$$

1.3 Probabilistic interpretation

Justify Least-squared regression by probability.
Assume that target variables and the inputs follows:

$$y^{(i)}=\theta^Tx^{(i)}+\epsilon^{(i)}$$

Further assume $\epsilon^{(i)}$, **the error term**, are **IID distributed** according to **Normal distribution**.

$$\epsilon^{(i)}\sim\mathcal{N}(0,\sigma^2)$$

$$p(\epsilon^{(i)})=\frac{1}{\sqrt{2\pi}\sigma}exp\left(\frac{-(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

$$\implies p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

Given X and θ, find distribution of the $y^{(i)}$, which uses the **Likelihood Function**

$$L(\theta) = L(\theta; X, \vec{y}) = p(\vec{y}|X;\theta)$$

Based on the independence assumption:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{n} p(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} exp\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \end{aligned}$$

Principle of maximum likelihood

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= n \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y^{(i)} - \theta^T x^{(i)})^2 \end{aligned}$$

Thus, same as minimizing $\frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \theta^T x^{(i)})^2$
Note that the choice of θ does NOT depend on σ (As exponential family nature...)

1.4 Locally weighted linear regression

Terminologies

- **Underfitting**: Data shows structure not captured by model.
- **Overfitting**: Too captured by model

Choice of features is important!
However, here in LWR, assume choice of features less critical. The procedure goes as:

1. Fit θ to minimize $\sum_i (y^{(i)} - \theta^T x^{(i)})^2$
2. Output $\theta^T x$

$w^{(i)}$: **weights**, a fairly choice is $exp\left(-\frac{(x^{(i)}-x)^2}{2\tau^2}\right)$ where weights depend on the particular point x, in this term, a **higher weight is given to training points close to** x. The parameter τ controls how quickly the weight falls off with distance, which is called **Bandwidth** parameter.

(If x is vector-valued, $w^{(i)} = exp\left(\frac{-(x^{(i)}-x)^T(x^{(i)}-x)}{2\tau^2}\right)$ or $exp\left(\frac{-(x^{(i)}-x)^T\Sigma^{-1}(x^{(i)}-x)}{2\tau^2}\right)$)

Parametric and Non-parametric

- **Parametric**: Fixed finite number of params, once set, no longer need to keep training set
- **Non-parametric**: Hypothesis grows linearly with size of training set. (Keep training set)