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:

$$\begin{aligned}&\text{Loop}\{\\&\quad\text{for } i = 1 \text{ to } n\{\\&\qquad\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}\\&\quad\}\\&\}\end{aligned}$$

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.