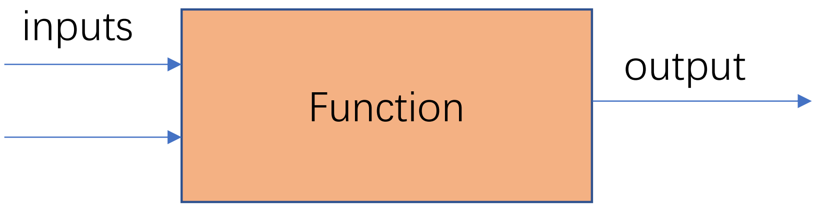
Machine Learning Part I

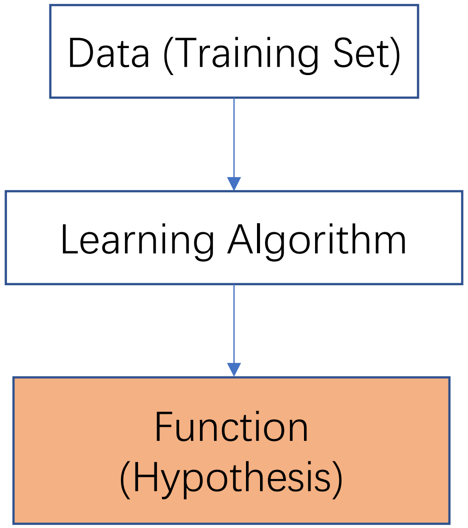
What is ML?

- **Learn a causal relationship (hypothesis / function) between two things (input variable and output variable) from experience (Training data).**

- The field of study that gives computers the ability to learn without being explicitly programmed.

Linear Regression



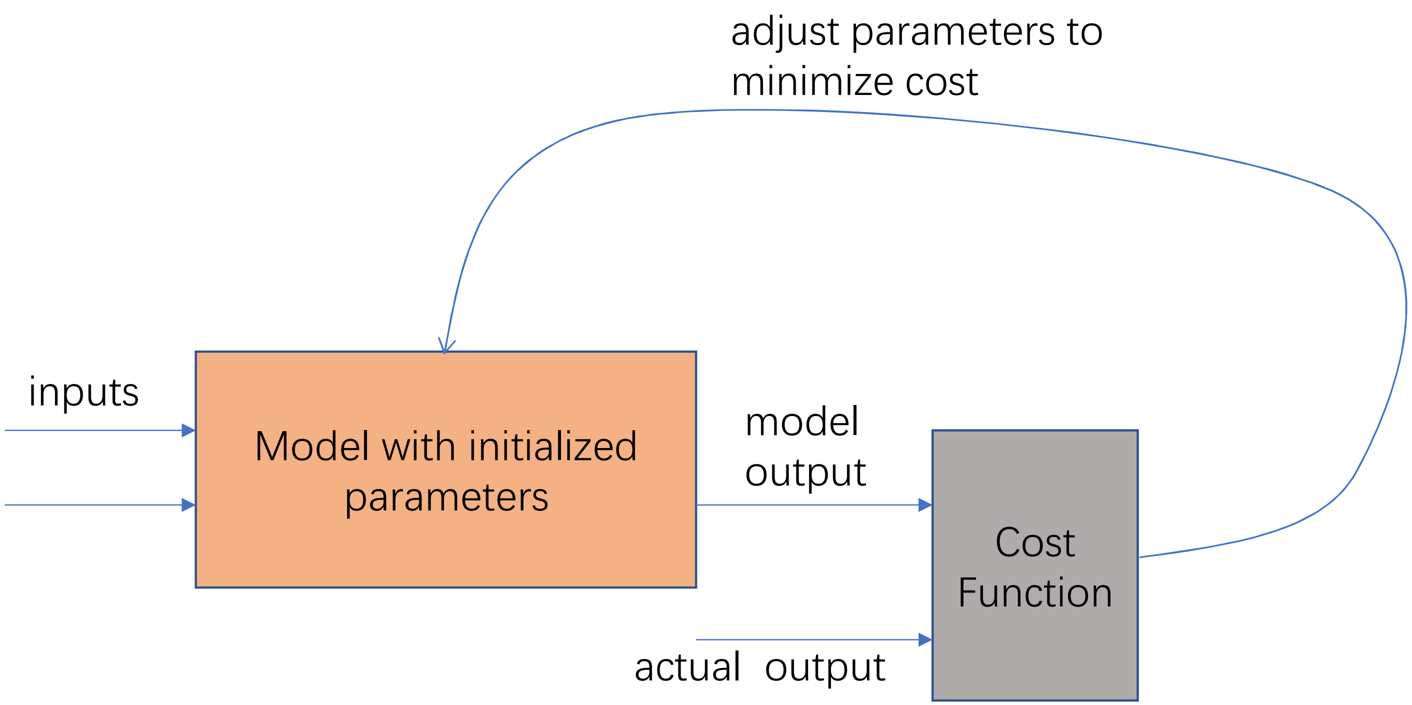


a simple example:

$h(x) = \theta_0 + \theta_1 x$

univariate linear regression

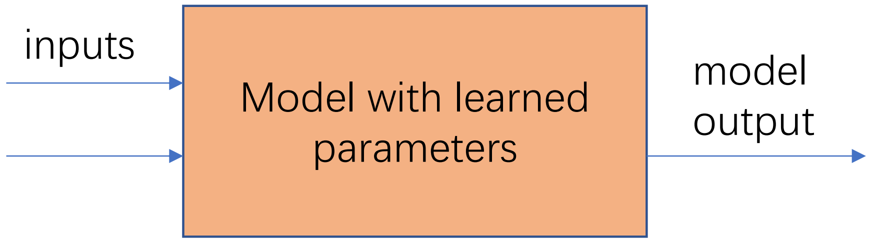
Learning Process



$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

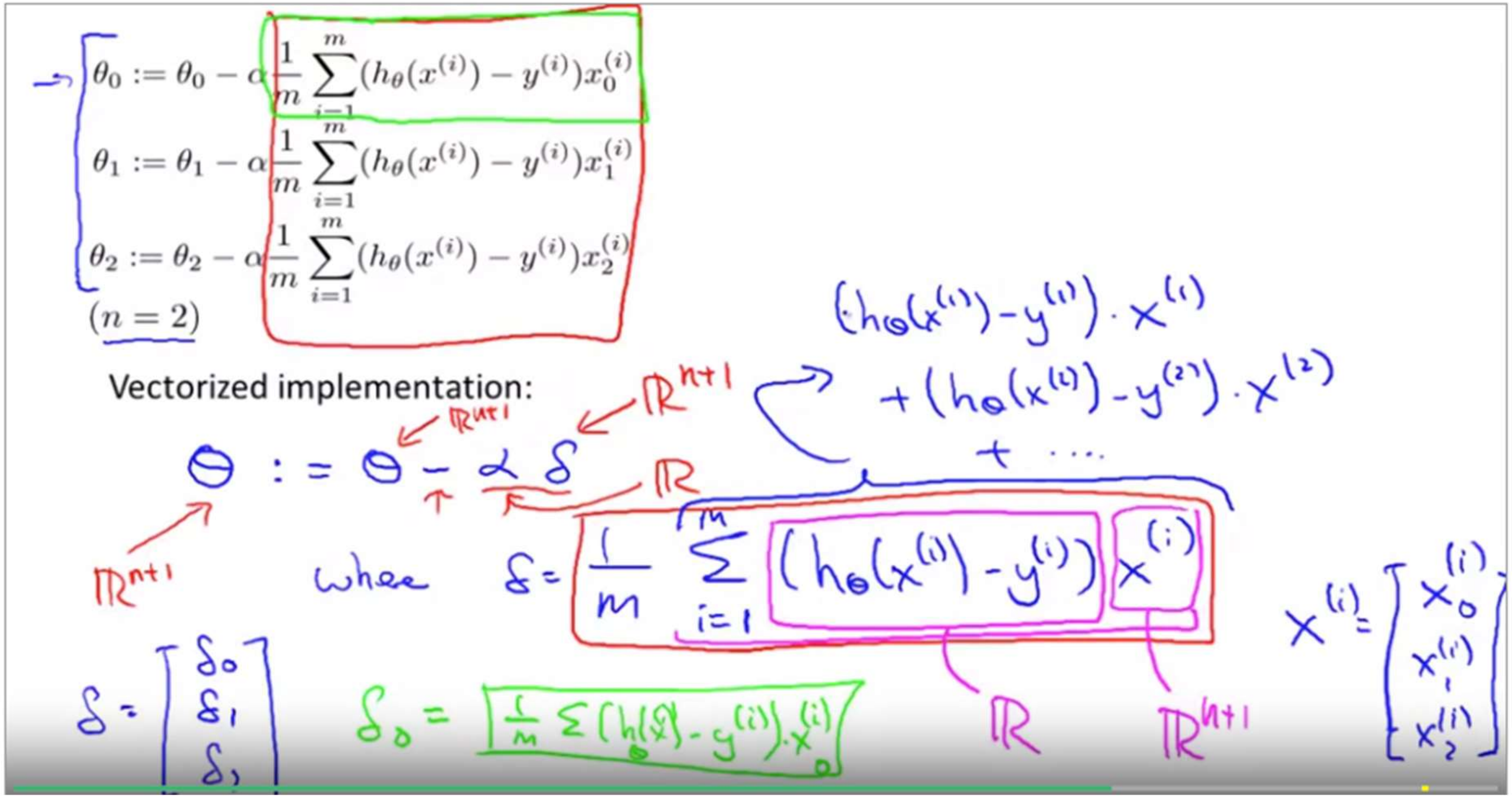
In the multivariate case, the cost function can also be written in the following vectorized form:

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


inputs
Model with learned parameters
model output

Gradient Descent

Vectorization



How to define your own ML problems

- Give hypothesis function including all input variables and the unknown parameters.

Using the definition of matrix multiplication, our multivariable hypothesis function can be concisely represented as:

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \dots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

- theta is the parameter vector
- x is the feature vector

Tricks in Gradient Descent

- Feature scaling: make sure features are on a **similar scale**
  - normalize
  - mean normalization

$$x_i := \frac{x_i - \mu_i}{s_i}$$

Where μ_i is the **average** of all the values for feature (i) and s_i is the range of values (max - min), or s_i is the standard deviation.

- **Choose a good learning rate:**

  use plot to make sure that $J(\theta)$ decreases after each iteration
  - if $J(\theta)$ increases, then perhaps α is too big (overshooting)
  - for sufficiently small α, $J(\theta)$ should decreases after each iteration
  - try 0.001, 0.003, 0.01, 0.03, 0.1,⋯ plot. Find one value that is too large
    Then choose a value slightly small than the largest value.

Features & polynomial regression

- With domain knowledge, we can use existing inputs to **design new features** that fit better to the specific problem

- when a hypothesis is polynomial function, we can still use the multivariate linear regression model.

For example, if our hypothesis function is $h_\theta(x) = \theta_0 + \theta_1 x_1$ then we can create additional features based on x_1, to get the quadratic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$ or the cubic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$

In the cubic version, we have created new features x_2 and x_3 where $x_2 = x_1^2$ and $x_3 = x_1^3$.

To make it a square root function, we could do: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$

Normal equation for linear regression

- When the features are not very large ($n < 1k$) , solve θ directly in one step instead of many steps iteratively.

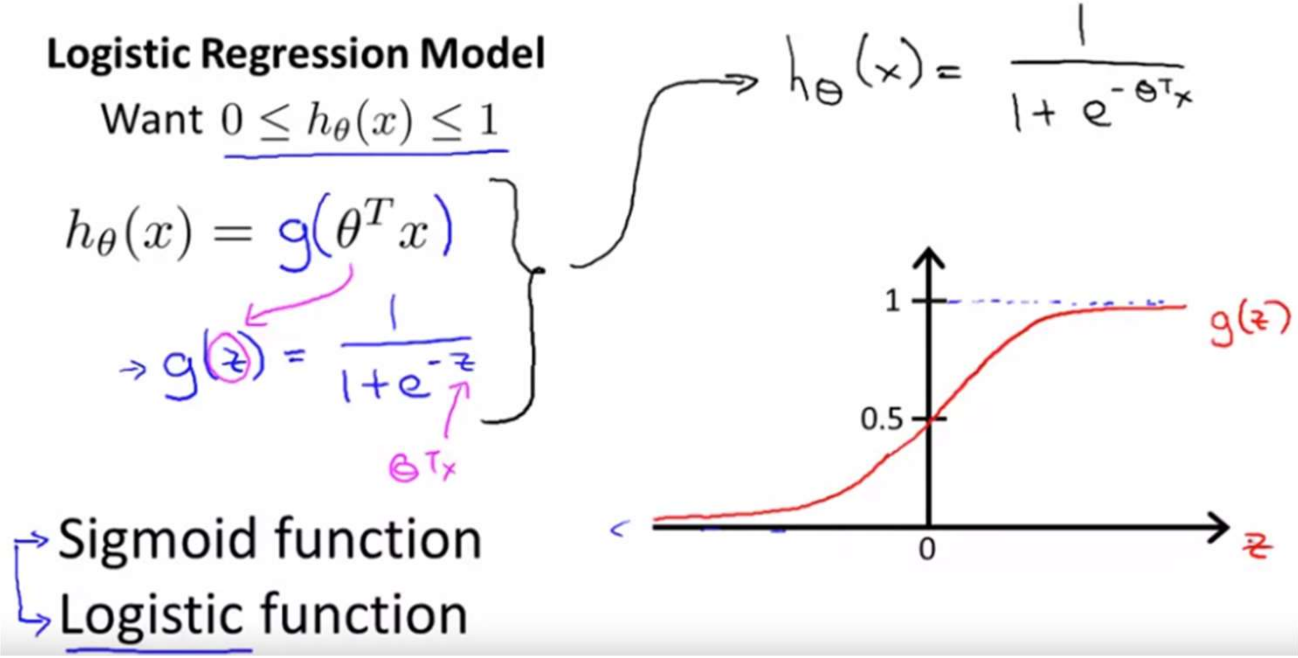
- However normal equation don't work for more sophisticated learning algorithm;

  while Gradient descent can still be used!

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

Logistic Regression/Classification

- Logistic regression: $0 < h_\theta\ (x) < 1$

→ 

$h(\theta) = P(y\ =\ 1|\ x;\ \theta)$ probability that y = 1, given x, parameterized by θ

Decision boundary

$$h_\theta(x) = g(\theta^T x) \geq 0.5$$
$$when\ \theta^T x \geq 0$$

From these statements we can now say:

$$\theta^T x \geq 0 \Rightarrow y = 1$$
$$\theta^T x < 0 \Rightarrow y = 0$$

The **decision boundary** is the line that separates the area where y = 0 and where y = 1. It is created by our hypothesis function.

Cost Function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

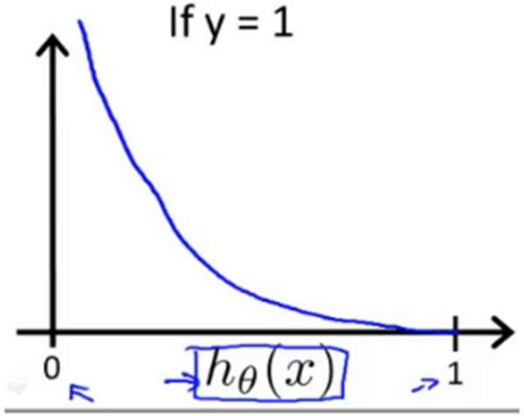
$$\mathrm{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if } \mathrm{y} = 1$$

$$\mathrm{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if } \mathrm{y} = 0$$

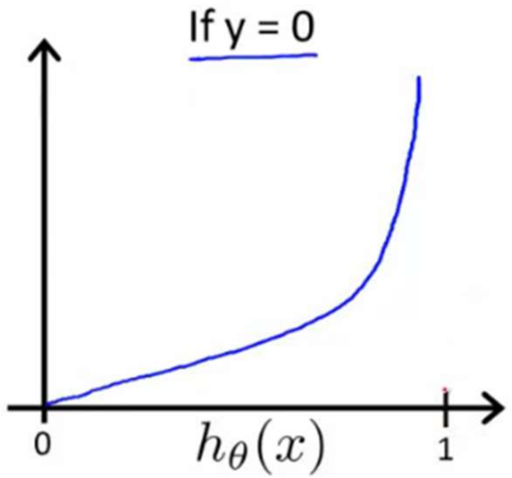
$$\mathrm{Cost}(h_\theta(x), y) = -y \;\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$$

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

When y = 1, we get the following plot for $J(\theta)$ vs $h_\theta(x)$:



Similarly, when y = 0, we get the following plot for $J(\theta)$ vs $h_\theta(x)$:



Gradient Descent

Remember that the general form of gradient descent is:

$$
\begin{aligned}
& Repeat \; \{ \\
& \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\
& \}
\end{aligned}
$$

We can work out the derivative part using calculus to get:

$$
\begin{aligned}
& Repeat \; \{ \\
& \theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \\
& \}
\end{aligned}
$$

Notice that this algorithm is identical to the one we used in linear regression. We still have to simultaneously update all values in theta.

A vectorized implementation is:

$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - \vec{y})$

Advanced Optimization

Given θ, we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial\theta_j}J(\theta)$ (for $j = 0, 1, \ldots, n$)

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

Advantages:

- No need to manually pick α
- Often faster than gradient descent.

Disadvantages:

- More complex

Feed $J(\theta)$ and gradient of $J(\theta)$ into other advanced optimization algorithms, e.g. fminunc

Example:

$\min_{\theta} J(\theta)$

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

$\theta_1 = 5, \theta_2 = 5.$

$$\boxed{J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2}$$

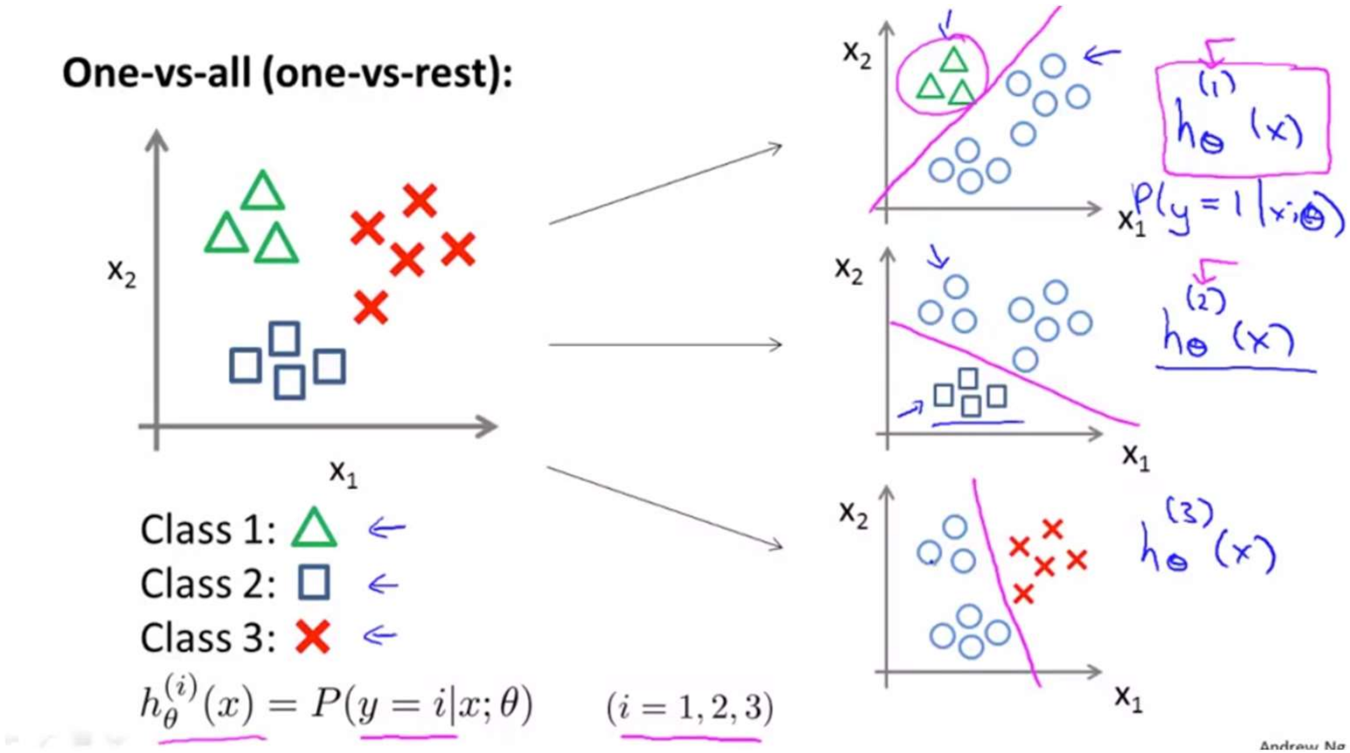
$$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$$

$$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$$

```
function [jVal, gradient]
            = costFunction(theta)
  jVal = (theta(1)-5)^2 + ...
          (theta(2)-5)^2;
  gradient = zeros(2,1);
  gradient(1) = 2*(theta(1)-5);
  gradient(2) = 2*(theta(2)-5);
```

```
options = optimset('GradObj', 'on', 'MaxIter', '100');
initialTheta = zeros(2,1);
[optTheta, functionVal, exitFlag] ...
     = fminunc(@costFunction, initialTheta, options);
```

Multiclass classification (one-vs-all)



One-vs-all

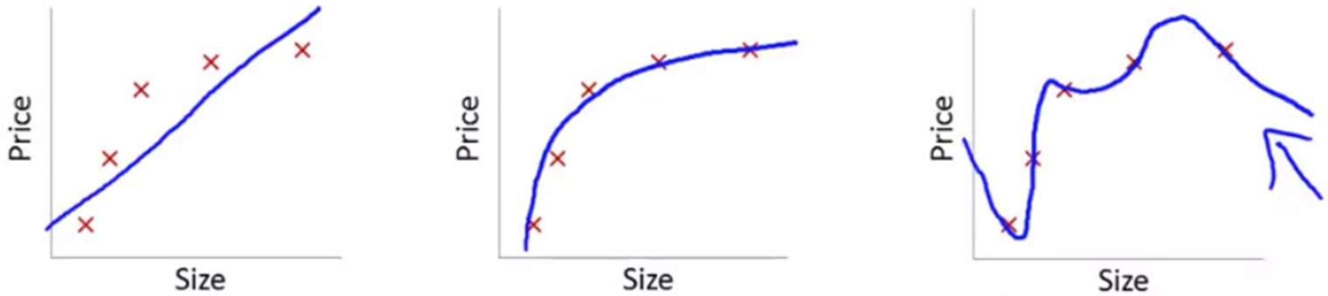
Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.

On a new input x, to make a prediction, pick the class i that maximizes

$$\max_i h_\theta^{(i)}(x)$$

The Problem of Overfitting

Example: Linear regression (housing prices)



$\rightarrow \theta_0 + \theta_1 x$ "Underfit" "High bias"

$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2$ "Just right"

$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$ "Overfit" "High variance"

Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

1) Reduce the number of features:

- Manually select which features to keep.
- Use a model selection algorithm (studied later in the course).

2) Regularization

- Keep all the features, but reduce the magnitude of parameters θ_j.
- Regularization works well when we have a lot of slightly useful features.

Regularization : Cost Function

- shrink parameter θ,

  to make the curve smoother and the hypothesis simpler,

  → solve overfitting

In regularized linear regression, we choose θ to minimize

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \underline{\lambda}\sum_{j=1}^{n}\theta_j^2\right]$$

What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?

However, a too big λ would lead to underfitting

Gradient Descent

We will modify our gradient descent function to separate out θ_0 from the rest of the parameters because we do not want to penalize θ_0.

$$
\begin{aligned}
&\text{Repeat } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} \\
&\quad \theta_j := \theta_j - \alpha \left[\left(\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right] \qquad\qquad j \in \{1, 2...n\} \\
&\}
\end{aligned}
$$

The term $\frac{\lambda}{m}\theta_j$ performs our regularization. With some manipulation our update rule can also be represented as:

$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$

The first term in the above equation, $1 - \alpha\frac{\lambda}{m}$ will always be less than 1. Intuitively you can see it as reducing the value of θ_j by some amount on every update. Notice that the second term is now exactly the same as it was before.

Solve theta directly:

$$\theta = \left(X^T X + \lambda \cdot L\right)^{-1} X^T y$$

$$\text{where } L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

Regularized Logistic Regression

Cost Function

Recall that our cost function for logistic regression was:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_\theta(x^{(i)})) + (1 - y^{(i)})\ \log(1 - h_\theta(x^{(i)}))]$$

We can regularize this equation by adding a term to the end:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_\theta(x^{(i)})) + (1 - y^{(i)})\ \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

The second sum, $\sum_{j=1}^{n}\theta_j^2$ **means to explicitly exclude** the bias term, θ_0. I.e. the θ vector is indexed from 0 to n (holding n+1 values, θ_0 through θ_n), and this sum explicitly skips θ_0, by running from 1 to n, skipping 0. Thus, when computing the equation, we should continuously update the two following equations:

Gradient descent

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha\underbrace{\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j\right]}_{\frac{\partial}{\partial\theta_j}J(\theta)} \qquad (j = \cancel{0}, \underbrace{1, 2, 3, \ldots, n}_{\theta_1 \ldots\ \theta_n})$$

$\}$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

Decision Tree

- Learn a hierarchy of if/else questions, leading to a decision