Learning From Data
Lecture 2: Linear Regression & Logistic Regression

Yang Li yangli@sz.tsinghua.edu.cn

9/27/2019

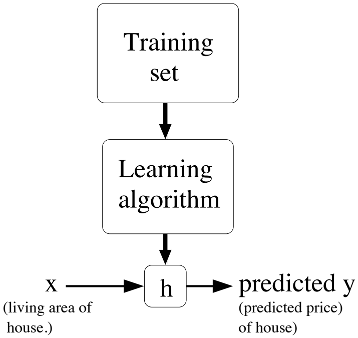
Today's Lecture

Supervised Learning (Part I)

- Linear Regression
- Binary Classification
- Multi-Class Classification

Review: Supervised Learning

- Input space: $\mathcal{X}$, Target space: $\mathcal{Y}$
- Given training examples, we want to learn a **hypothesis** function $h : \mathcal{X} \rightarrow \mathcal{Y}$ so that $h(x)$ is a "good" predictor for the corresponding y.



- y is discrete (categorical): **classification problem**
- y is continuous (real value): **regression problem**

Review: Inference vs Learning

Given training data of x and y,

Inference

knowing the structure of f, find good models to describe f. i.e. model the data generation process $\leftarrow$ focus of statistics

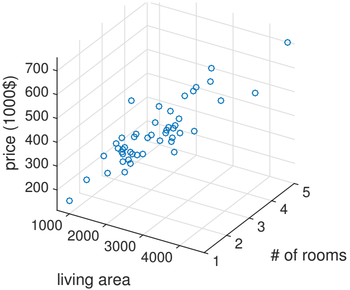
Prediction

given **future** data samples of x, predict the corresponding output data y using f. $\leftarrow$ focus of machine learning

Linear Regression

Example: predict Portland housing price

Living area (ft^2)	# bedrooms	Price ($1000)
x_1	x_2	y
2104	3	400
1600	3	330
2400	3	369
⋮	⋮	⋮



Linear Approximation

A linear model

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

θ_i's are called **parameters**.

Using vector notation,

$$h(x) = \theta^T x, \quad \text{where } \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

Alternative Notation

$$h(x) = w_1 x_1 + w_2 x_2 + b$$

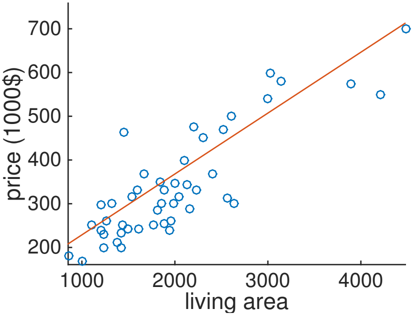
w_1, w_2 are called **weights**, b is called the **bias**

$$h(x) = w^T x + b, \quad \text{where } w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Apply model to new data

Suppose the optimal parameters θ is known, make a prediction given feature x:

$$\hat{y} = h_\theta(x) = \theta^T x$$



$$\theta_0 = 89.60, \theta_1 = 0.1392, \theta_2 = -8.738$$

Model Estimation

How to estimate model parameters θ (or w and b) from data?

Least Square Estimation

Minimize sum of the prediction error squared (least square error) with respect to θ

Maximum Likelihood Estimation

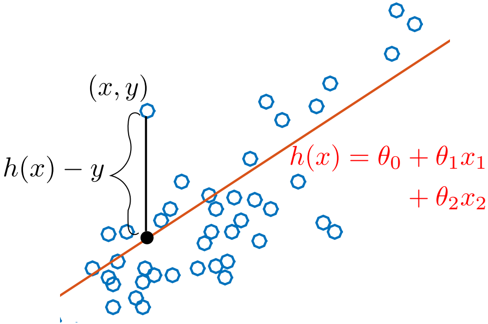
- Assume the data are generated from $h(x)$ with some noise distribution.
- Determines the parameters θ most likely to produce the observed data.

Other estimation methods exist, e.g. Bayesian estimation

Ordinary Least Square

Cost function:

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



- This model is called **ordinary least square**

Ordinary Least Square

Cost function:

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

- This model is called **ordinary least square**

Ordinary Least square problem

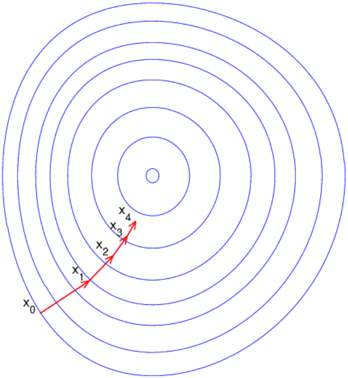
$$\begin{aligned} &\min_{\theta} J(\theta) \\ = &\min_{\theta} \frac{1}{2}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2 \end{aligned}$$

How to minimize $J(\theta)$?

- Numerical solution: gradient descent, Newton's method
- Analytical solution: normal equation

Gradient descent

A first-order iterative optimization algorithm for finding the minimum of a function $J(\theta)$.



Key idea

Start at an initial guess, repeatedly change θ to decrease $J(\theta)$:

$$\theta := \theta - \alpha \nabla J(\theta)$$

α is the **learning rate**

Theorem

If $J(\theta)$ is convex, gradient descent finds the global minimum.

For the ordinary least square problem,
$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2 = \frac{1}{2}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2$,

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}, \text{ where } \frac{\partial J(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j}\left[\frac{1}{2}\sum_{i=1}^{m}\left(\theta^T x^{(i)} - y^{(i)}\right)^2\right]$$

$$= \sum_{i=1}^{m}\left(\theta^T x^{(i)} - y^{(i)}\right) x_j^{(i)}$$

Gradient descent for ordinary least square

Cost function: $\nabla J(\theta) = \sum_{i=1}^{m} \left(\theta^T x^{(i)} - y^{(i)}\right) x_j^{(i)}$
Gradient descent update: $\theta := \theta - \alpha \nabla J(\theta)$

Batch Gradient Descent

```
Repeat until convergence{
    θj = θj + α Σ_{i=1}^{m} (y^(i) − hθ(x^(i))) xj^(i) for every j
}
```

$$\theta_j = \theta_j + \alpha \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} \quad \text{for every } j$$

θ is only updated after we have seen all m training samples.

Batch gradient descent

```
Repeat until convergence{
  θ_j = θ_j + α ∑_{i=1}^{m} (y^(i) − h_θ(x^(i))) x_j^(i) for every j
}
```

$$\text{Repeat until convergence}\{\quad \theta_j = \theta_j + \alpha \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} \ \text{for every j} \quad\}$$

Stochastic gradient descent

```
Repeat until convergence{
  for i = 1...m {
    θ_j = θ_j + α(y^(i) − h_θ(x^(i))) x_j^(i) for every j
  }
}
```

θ is updated each time a training example is read

- Stochastic gradient descent gets θ close to minimum much faster
- Good for regression on large data

Minimize $J(\theta)$ Analytically

The matrix notation

$$X = \begin{bmatrix} - \left(x^{(1)}\right)^T - \\ - \left(x^{(2)}\right)^T - \\ \vdots \\ - \left(x^{(m)}\right)^T - \end{bmatrix}, \quad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

X is called the **design matrix**. The least square function can be written as

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

Compute the gradient of $J(\theta)$:

$$\begin{aligned}\nabla_\theta J(\theta) =& \nabla_\theta \left[\frac{1}{2}(X\theta - y)^T(X\theta - y)\right]\\ =& X^TX\theta - X^Ty\end{aligned}$$

Since $J(\theta)$ is **convex**, x is a global minimum of $J(\theta)$ when $\nabla J(\theta) = 0$.

The Normal equation

$$\theta = (X^TX)^{-1}X^Ty$$

$(X^TX)^{-1}X^T$ is called the **Moore-Penrose pseudoinverse of** X

Which method to use?

gradient descent	normal equation
iterative solution	exact solution
need to choose proper learning parameter α for cost function to converge	numerically unstable when X is ill-conditioned. e.g. features are highly correlated
works well for large number of samples m	solving equation is slow when m is large

Minimize $J(\theta)$ using Newton's Method

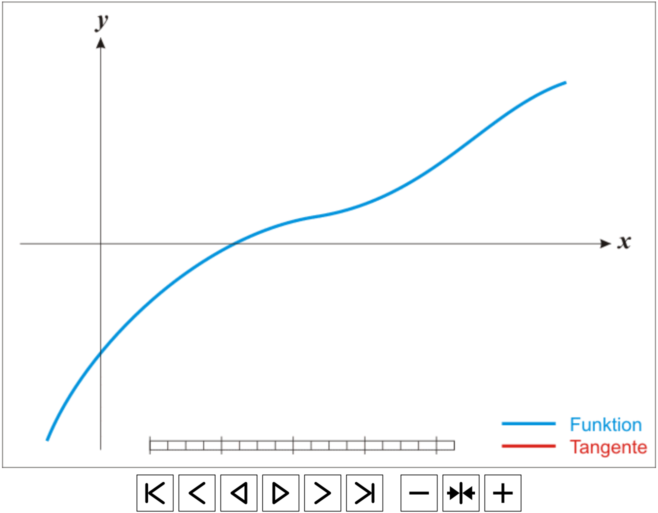
Newton's method solves real functions $f(x) = 0$ by iterative approximation

- Update rule: $x_{n+1} := x_n - \frac{f(x_n)}{f'(x_n)}$

Geometric intuition of Newton's method

- Find tangent line of f at (x_n, y_n)
- $x_{n+1} \leftarrow$ x-intercept of the tangent line
- $y_{n+1} \leftarrow f(x_{n+1})$

Newton's Method Demo



https://en.wikipedia.org/wiki/File:NewtonIteration_Ani.gif

Minimize $J(\theta)$ using Newton's Method

Newton's method for optimization $\min_\theta J(\theta)$

Use newton's method to solve $\nabla_\theta J(\theta) = 0$:

- x is one-dimensional:

$$\theta := \theta - \frac{f'(x)}{f''(x)}$$

- x is multidimensional:

$$\theta = \theta - H^{-1}(\theta)\nabla J(\theta)$$

  where H is the Hessian matrix of $J(\theta)$.

a.k.a Newton-Raphson method

Newton's Method for Optimization

```
Initialize θ
While θ has not coverged {
  θ := θ − H^{-1}(θ)∇J(θ)
}
```

Performance of Newton's method:

- Needs fewer interations than batch gradient descent
- Computing H^{-1} is time consuming
- Faster in practice when n is small

Maximum Likelihood Estimation

Consider target y is modeled as

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

and suppose $\epsilon^{(i)}$ are *independently and identically distributed (IID)* to Gaussian distribution $\mathcal{N}(0, \sigma^2)$, then

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

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

Maximum Likelihood Estimation

The **likelihood** of this model with respect to θ is

$$L(\theta) = p(\vec{y}|X;\theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta)$$

Maximum likelihood estimation of θ:

$$\theta_{MLE} = \underset{\theta}{\operatorname{argmax}}\, L(\theta)$$

Maximum Likelihood Estimation

We compute log likelihood,

$$\log L(\theta) = \log \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) = \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)};\theta)$$

$$= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y^{(i)} - \theta^T x)^2}{2\sigma^2}\right)$$

$$= m \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2$$

Then $\operatorname{argmax}_\theta \log L(\theta) \equiv \operatorname{argmin}_\theta \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2$.

Under the assumptions on $\epsilon^{(i)}$, least-squares regression corresponds to the maximum likelihood estimate of θ.

Linear Regression Summary

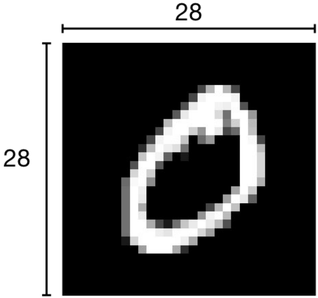
- Least square regression
- Solving least square:
  - gradient descent
  - normal equation
  - newton's method
- Probabilistic interpretation: maximum likelihood

A binary classification problem

Classify binary digits

- Training data: 12600 grayscale images of handwritten digits

- Each image is represent by a vector $x^{(i)}$ of dimension $28 \times 28 = 784$
- Vectors $x^{(i)}$ are normalized to [0,1]



Binary classification: $\mathcal{Y} = \{0, 1\}$

- negative class: $y^{(i)} = 0$
- positive class: $y^{(i)} = 1$

Logistic Regression Hypothesis Function

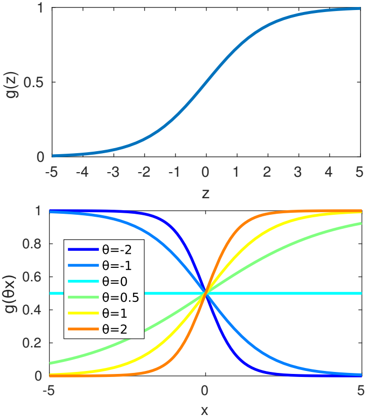
Sigmoid function

$$g(z) = \frac{1}{1 + e^{-z}}$$

- $g : \mathbb{R} \to (0, 1)$
- $g'(z) = g(z)(1 - g(z))$

Hypothesis function for logistic regression:

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



Maximum likelihood estimation for logistic regression

Logistic regression assumes $y|x$ is **Bernoulli distributed**.
e.g. tossing a coin with $p(head) = h_\theta(x)$

$$p(y \mid x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$

- $p(y = 1 \mid x; \theta) = h_\theta(x)$
- $p(y = 0 \mid x; \theta) = 1 - h_\theta(x)$

Given m independently generated training examples, the likelihood function is:

$$L(\theta) = p(\vec{y}|X; \theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; \theta)$$

$$l(\theta) = \log(L(\theta)) = \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

$l(\theta)$ is concave!

Maximum likelihood estimation for logistic regression

Solve $\text{argmax}_\theta \, l(\theta)$ using gradient ascent:

$$\frac{\partial l(\theta)}{\partial \theta_j} = \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

Stocastic Gradient Ascent

```
Repeat until convergence{
  for i = 1...m {
    θ_j = θ_j + α(y^(i) − h_θ(x^(i)))x_j^(i)   for every j
  }
}
```

- Update rule has the same form as least square regression, but with different hypothesis function h_θ

Binary Digit Classification

Using the learned classifier

Given an image x, the predicted label is

$$\hat{y} = \begin{cases} 1 & g(\theta^T x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Binary digit classification results

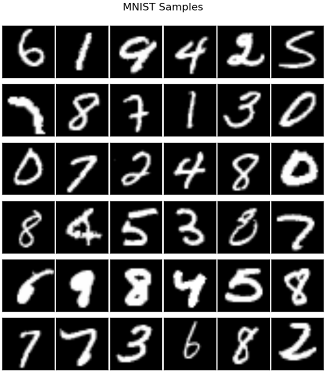
	sample size	accuracy
Training	16200	100%
Testing	1225	100%

- Testing accuracy is 100% since this problem is relatively easy.

Multi-class classification

Each data sample belong to one of $k > 2$ different classes.

$$\mathcal{Y} = \{1, \ldots, k\}$$



Given new sample $x \in \mathbb{R}^k$, predict which class it belongs.

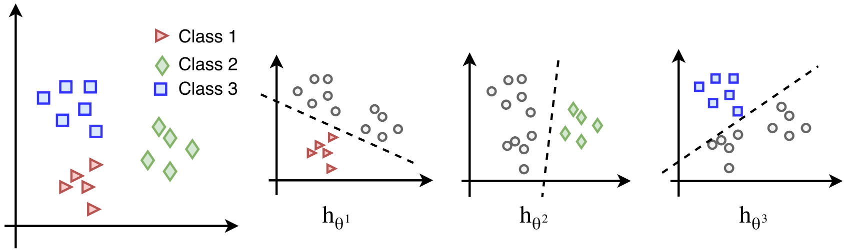
Naive Approach: Convert to binary classification

One-Vs-Rest

Learn k classifiers $h_1, \ldots, h_k$. Each h_i classify one class against the rest of the classes.

Given a new data sample x, its predicted label $\hat{y}$:

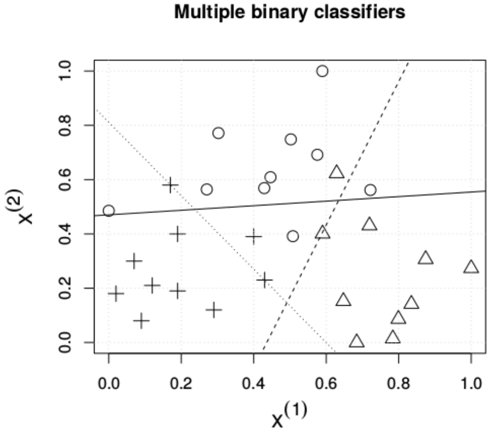
$$\hat{y} = \underset{i}{\operatorname{argmax}}\, h_i(x)$$



Multiple binary classifiers

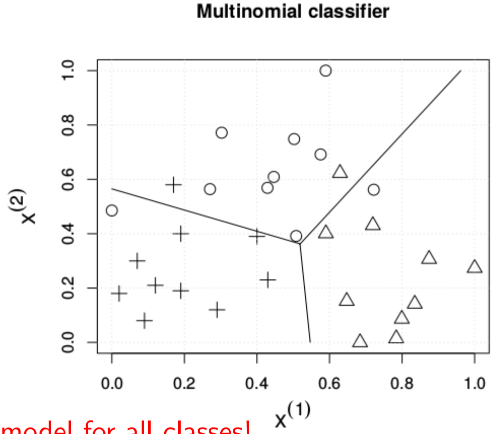
Drawbacks of One-Vs-Rest:

- Class unbalance: more negative samples than positive samples
- Different classifiers may have different confidence scales



Drawbacks of One-Vs-Rest:

- Class imbalance: more negative samples than positive samples
- Different classifiers may have different confidence scales



Learn one model for all classes!

Extend logistic regression: Softmax Regression

Assume $p(y|x)$ is **multinomial distributed**.
e.g. outcomes of rolling a k-sided die n times, each side has independent probability $\phi_1, \dots \phi_k$

Hypothesis function for sample x:

$$h_\theta(x) = \begin{bmatrix} p(y=1|x;\theta) \\ \vdots \\ p(y=k|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x_j}} \begin{bmatrix} e^{\theta_1^T x} \\ \vdots \\ e^{\theta_k^T x} \end{bmatrix} = \text{softmax}(\theta^T x)$$

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{(z_j)}}$$

Parameters: $\theta = \begin{bmatrix} - & \theta_1^T & - \\ & \vdots & \\ - & \theta_k^T & - \end{bmatrix}$

Softmax Regression

Given $(x^{(i)}, y^{(i)}), i = 1, \ldots, m$, the log-likelihood of the Softmax model is

$$\begin{aligned}
\ell(\theta) &= \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}; \theta) \\
&= \sum_{i=1}^{m} \log \prod_{l=1}^{k} p(y^{(i)} = l|x^{(i)})^{\mathbf{1}\{y^{(i)}=l\}} \\
&= \sum_{i=1}^{m} \sum_{l=1}^{k} \mathbf{1}\{y^{(i)} = l\} \log p(y^{(i)} = l|x^{(i)}) \\
&= \sum_{i=1}^{m} \sum_{l=1}^{k} \mathbf{1}\{y^{(i)} = l\} \log \frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}}
\end{aligned}$$

Softmax Regression

Derive the stochastic gradient descent update:

- Find $\nabla_{\theta_l} \ell(\theta)$

$$\nabla_{\theta_l} \ell(\theta) = \sum_{i=1}^{m} \left[\left(\mathbf{1}\{y^{(i)} = l\} - P\left(y^{(i)} = l | x^{(i)}; \theta \right) \right) x^{(i)} \right]$$

Property of Softmax Regression

- Parameters $\theta_1, \ldots \theta_k$ are not independent: $\sum_j p(y = j|x) = \sum_j \phi_j = 1$
- Knowning $k - 1$ parameters completely determines model.

Invariant to scalar addition

$$p(y|x;\theta) = p(y|x;\theta - \psi)$$

Proof.

Relationship with Logistic Regression

When K = 2,

$$h_\theta(x) = \frac{1}{e^{\theta_1^T x} + e^{\theta_2^T x}} \begin{bmatrix} e^{\theta_1^T x} \\ e^{\theta_2^T x} \end{bmatrix}$$

Replace $\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ with $\theta - \theta_2 = \begin{bmatrix} \theta_1 - \theta_2 \\ 0 \end{bmatrix}$,

$$\begin{aligned} h_\theta(x) &= \frac{1}{e^{\theta_1^T - \theta_2^T x} + e^{0x}} \begin{bmatrix} e^{(\theta_1 - \theta_2)^T x} \\ e^{0^T x} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{1+e^{(\theta_1-\theta_2)^T x}} \\ 1 - \frac{1}{1+e^{(\theta_1-\theta_2)^T x}} \end{bmatrix} = \begin{bmatrix} g(\theta^T x) \\ 1 - g(\theta^T x) \end{bmatrix} \end{aligned}$$

When to use Softmax?

- When classes are mutually exclusive: use Softmax
- Not mutually exclusive: multiple binary classifiers may be better