# CPE/EE 695: Applied Machine Learning

## *Lecture 3-1: Logistic Regression*

Dr. Shucheng Yu, Associate Professor
Department of Electrical and Computer Engineering
Stevens Institute of Technology

# Logistic Regression

Usually be used for binary **classification:**

**Pr(Y|X)**, where Y is a binary variable.  (why probability?)

# Logistic Regression

Usually be used for binary **classification:**

**Pr(Y|X)**, where Y is a binary variable. (why probability?)

E.g., Pr(Tomorrow snow | today windy)

# Logistic Regression

Usually be used for binary **classification:**

**Pr(Y|X)**, where Y is a binary variable. (why probability?)

E.g., Pr(Tomorrow snow | today windy)

Let Pr( Y=1 | X=x) = $p(x;\theta)$

# Logistic Regression

Usually be used for binary **classification:**

**Pr(Y|X)**, where Y is a binary variable. (why probability?)

E.g., Pr(Tomorrow snow | today windy)

Let Pr( Y=1 | X=x) = $p(x;\theta)$

Maximize likelihood:

$$\prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i}$$

# Logistic Regression

Usually be used for binary **classification:**

**Pr(Y|X)**, where Y is a binary variable. (why probability?)

E.g., Pr(Tomorrow snow | today windy)

Let Pr( Y=1 | X=x) = $p(x;\theta)$

**Assumption**: p is modeled with parameter $\theta$; otherwise, optimization problem doesn't work

Maximize likelihood:

$$\prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i}$$

# Logistic Regression

Let Pr( Y=1 | X=x) = $p(x;\theta)$

Maximize likelihood:

$$\prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i}$$

# Logistic Regression

Let Pr( Y=1 | X=x) = $p(x;\theta)$

Maximize likelihood:

$$\prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i}$$

Task: to estimate $\theta$ by maximizing the likelihood.

# Logistic Regression

Let Pr( Y=1 | X=x) = $p(x;\theta)$

Maximize likelihood:

$$\prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i}$$

Task: to estimate $\theta$ by maximizing the likelihood.

How can we use **linear regression** to solve this?

# Logistic Regression

Let Pr( Y=1 | X=x) = $p(x;\theta)$

Task: to estimate $\theta$ by maximizing the likelihood.

How can we use **linear regression** to solve this?

Attempt 1: assume $p(x;\theta)$ be a linear function of $x$

Attempt 2: assume $\log p(x;\theta)$ be a linear function of $x$

Attempt 3: assume $\log \frac{p}{1-p}$ be a linear function of $x$ (good)

Remember: 0<= p <= 1

# Logistic Regression

Logistic regression model

$$\log \frac{p}{1-p} = \theta_0 + x \cdot \theta$$

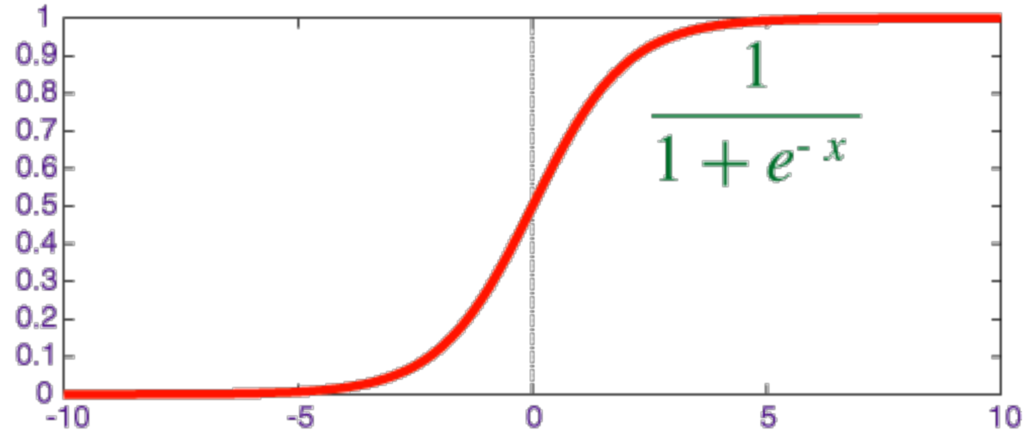
Which gives $p(x) = \frac{1}{1+ e^{-\theta_0 + x \cdot \theta}}$

# Logistic Regression

Logistic Regression model estimated probability:

$$\hat{p} = h_w(x) = \sigma(\theta^T \cdot x)$$

where $\sigma(\cdot)$ is a logistic function (or sigmoid function).




Prediction:

$$\hat{y} = \begin{cases} 0, & if\ \hat{p} < 0.5 \\ 1, & if\ \hat{p} \geq 0.5 \end{cases}$$

Training the logistic regression model $\hat{p}(x) = \sigma(\theta^T \cdot x)$ is to learn the best value of parameter $\theta$ that makes the model fit the training data.

# Logistic Regression

To train a logistic regression model, we first need to define a performance measure. A commonly used measure is so-called the **log loss** function:

$$\mathbf{J}(\boldsymbol{\theta}) = -\frac{\mathbf{1}}{\boldsymbol{m}}\sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{m}}[\boldsymbol{y}^{(\boldsymbol{i})}\mathbf{log}(\widehat{\boldsymbol{p}}^{(\boldsymbol{i})}) + \left(\mathbf{1} - \boldsymbol{y}^{(\boldsymbol{i})}\right)\mathbf{log}(\mathbf{1} - \widehat{\boldsymbol{p}}^{(\boldsymbol{i})}))]$$

It is easier to explain the log loss function with one train example case, in which we want to maximize the posterior probability

$$\mathrm{P}(y|x) = \widehat{\boldsymbol{p}}^{\boldsymbol{y}}(\mathbf{1} - \widehat{\boldsymbol{p}})^{(\mathbf{1}-\boldsymbol{y})} = \begin{cases} \hat{p}, & when\ y = 1 \\ 1 - \hat{p}, & when\ y = 0 \end{cases}$$

Take log of both sides, we have $\mathbf{log}\boldsymbol{P}(\boldsymbol{y}|\boldsymbol{x}) = \boldsymbol{ylog}\widehat{\boldsymbol{p}} + (\mathbf{1} - \boldsymbol{y})\boldsymbol{log}(\mathbf{1} - \widehat{\boldsymbol{p}})$.

Average the sum of $m$ training examples, we obtain $J(\theta)$.

# Logistic Regression

Learning the logistic regression model is to find:

$$\hat{\theta} = \ argmin_{\theta}\, \mathrm{J}(\theta).$$

Where $\mathrm{J}(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(\hat{\boldsymbol{p}}^{(i)}) + \left(1 - y^{(i)}\right)\log(1 - \hat{\boldsymbol{p}}^{(i)}))],$

$\hat{p} = \sigma(\theta^{T} \cdot x)$

No Normal Equation (i.e., closed form solution) for θ.

But the cost function $\boldsymbol{J(\theta)}$ is **convex** and **derivable**. **Gradient Descent** is guaranteed to find global maximum.

# Training Logistic Regression Model

The gradient of the log loss function $J(\theta)$ is:

$$\boldsymbol{\nabla_{\theta} J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\delta\left(\theta^{T}\cdot x^{(i)}\right)-y^{(i)}\right)x_{j}^{(i)}}$$

At each round of GD, $\theta$ is updated as following (similar to linear regression, different values of $m$ for different modes of GD):

$$\theta = \theta - \eta \nabla_{\theta} J(\theta)$$

# Training Logistic Regression Model

Overfitting may also happen in logistic regression.

Similarly, to combat overfitting we can introduce a ***regularization*** term to the cost function $J(\theta)$:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(\hat{p}^{(i)}) + \left(1 - y^{(i)}\right)log(1 - \hat{p}^{(i)}))] + \boldsymbol{\lambda R(\theta)}$$

where $\lambda$ is a hyperparameter and $R(\theta)$ can be $\ell_k$-norm of $\theta$, i.e.,

$$R(\theta) = \parallel \theta \parallel_k = (\sum \theta_i{}^k)^{\frac{1}{k}}$$

Note: $R(\theta)$ is $\ell_2$-norm in Ridge regression, $\ell_1$-norm in Lasso regression.

# Multi-Class Classification

## **One-Vs-Rest** Method:

We can use **binary classifier** for multi-class classification with so-called the One-Vs-Rest (OvR) method. Specifically, it uses multiple rounds of binary classification for multi-class classification.

For example, to determine if an object X is a dog, cat or fish, we call a binary classifier *f()* as follows:

```
if f(X) outputs dog
    return dog;
else if f(X) outputs cat
    return cat;
else return fish
```

# Multi-Class Classification

## **Multinomial Logistic Regression**:

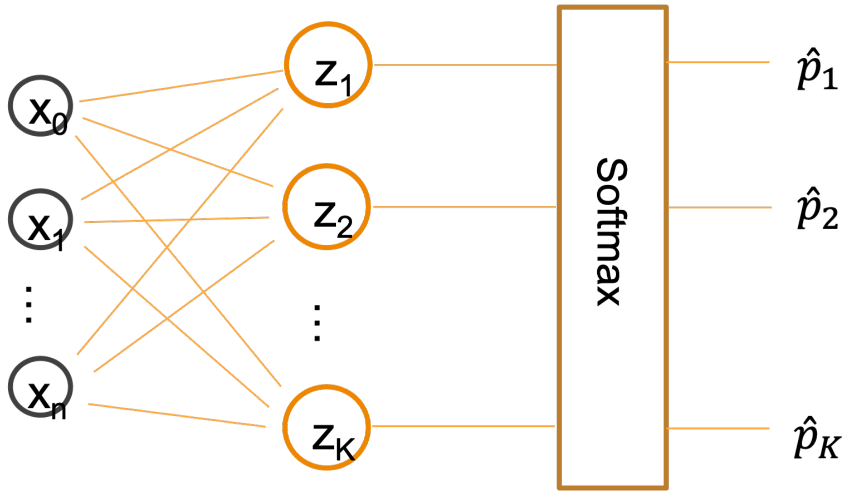
Another approach for multi-class classification is to use the multinomial logistic regression. For each class $1 \le k \le K$,

1) first compute $z_k(x) = \theta_k{}^T \cdot x$

2) then compute Softmax function:

$$\hat{p}_k = \delta(z_k(x))_k = \frac{\exp(z_k(x))}{\sum_{j=1}^{K} \exp(z_j(x))}$$

where $\theta_k$ is the vector of parameters of input features for $z_k$.



# Multi-Class Classification

## **Training Multinomial Logistic Regression Model**:

The performance measure is the **cross-entropy** cost function:

$$\boldsymbol{J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k{}^{(i)} log(\hat{p}_k{}^{(i)})}$$

where $y_k{}^{(i)} = \begin{cases} 1, & the\ i^{th}\ example\ of\ class\ k \\ 0, & i^{th}\ example\ not\ of\ class\ k \end{cases}$

GD can be used to train the multinomial logistic regression model. The gradient is:

$$\boldsymbol{\nabla_{\theta_k} J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\hat{p}_k{}^{(i)} - y_k{}^{(i)})x^{(i)}}$$

# Multi-Class Classification

## Other Approaches: **One-Vs-One Method**:

The One-Vs-One (OvO) method constructs a **binary classifier** for each pair of classes. Therefore, with ***K*** classes, we need to construct ***K(K-1)/2*** binary classifiers.

The decision at prediction time can be made by **counting the votes** from individual binary classifiers. In case of a tie, it compares the aggregated classification confidence (i.e., the output probability) of individual binary classifiers of each class and the higher one is selected.

The OvO method is **<u>slower</u>** than OvR. But for some algorithms (e.g., Kernel algorithms) which cannot scale with many training examples, this algorithm can be helpful.

# Multi-Class Classification

## Other Approaches: **Error-Correcting Output Codes**

The Error-Correcting Output Codes (ECOC) method encodes ***K* classes** into ***N* bit vectors**. Each class is represented as a bit in each bit vector. ECOC trains *N* **binary classifiers**, each splitting one group of classes from another (using the column bit vectors below). At prediction time, the N binary classifiers are called, the outputs of them yielding an N-bit vector. A class with the **closest Euclidean distance** to the **N-bit vector** is selected. To reduce the classification error, error correcting codes are used when generating the “code book”

| Class | Code Word | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $f_0$ | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 4 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 6 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 7 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 8 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 9 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |

A code book with K=9, N=15

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

**stevens.edu**