Logistic Regression

Instructor: Ping Li

Department of Statistics and Biostatistics
Department of Computer Science

Rutgers University

2015
Calculus Review: Derivatives

Simple derivatives:

\[
[\log x]' = \frac{1}{x}, \quad [x^n]' = nx^{n-1}, \quad [e^x]' = e^x, \quad [a^x]' = a^x \log a
\]

Chain rule:

\[
[f(h(x))]' = f'(h(x)) h'(x)
\]

\[
[\log (ax^2 + e^{2x})]' = \frac{1}{(ax^2 + e^{2x})} [ax^2 + e^{2x}]' = \frac{2ax + 2e^{2x}}{(ax^2 + e^x)}
\]

Multivariate derivatives:

\[
f(x, y) = a^x + x^ny + cy^2,
\]

\[
\frac{\partial f(x, y)}{\partial x} = a^x \log a + nx^{n-1}y, \quad \frac{\partial f(x, y)}{\partial y} = x^n + 2cy
\]
Quick Review of Numerical Optimization

Slides 4 - 15 are for reviewing some basic stuff about numerical optimization, which is essential in modern applications.
Maximum Likelihood Estimation (MLE)

Observations $x_i, i = 1$ to $n$, are i.i.d. samples from a distribution with probability density function $f_X(x; \theta_1, \theta_2, ..., \theta_k)$, where $\theta_j, j = 1$ to $k$, are parameters to be estimated.

The maximum likelihood estimator seeks the $\theta$ to maximize the joint likelihood

$$\hat{\theta} = \arg\max_{\theta} \prod_{i=1}^{n} f_X(x_i; \theta)$$

Or, equivalently, to maximize the log joint likelihood

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \log f_X(x_i; \theta)$$

This is a \textbf{convex} optimization if $f_X$ is \textit{concave} or \textit{-log-convex}.
An Example: Normal Distribution

If $X \sim N(\mu, \sigma^2)$, then $f_X(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi} \sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

Fix $\sigma^2 = 1$, $x = 0$. $f_X(x; \mu, \sigma^2)$

$log f_X(x; \mu, \sigma^2)$

It is Not concave, but it is a -log convex, i.e., a unique MLE solution exists.
Example of Exact MLE Solution

Given $n$ i.i.d. samples, $x_i \sim N(\mu, \sigma^2)$, $i = 1$ to $n$.

$$l(x_1, x_2, \ldots, x_n; \mu, \sigma^2) = \sum_{i=1}^{n} \log f_X(x_i; \mu, \sigma^2)$$

$$= - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2 - \frac{1}{2} n \log(2\pi\sigma^2)$$

$$\frac{\partial l}{\partial \mu} = \frac{1}{2\sigma^2} 2 \sum_{i=1}^{n} (x_i - \mu) = 0 \implies \hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

$$\frac{\partial l}{\partial \sigma^2} = \frac{1}{2\sigma^4} \sum_{i=1}^{n} (x_i - \mu)^2 - \frac{n}{2\sigma^2} = 0 \implies \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})^2.$$
Convex Functions

A function $f(x)$ is convex if the second derivative $f''(x) \geq 0$.

$f(x) = x^2 \implies f'' = 2 > 0$, i.e., $f(x) = x^2$ is convex for all $x$.  
$f(x) = x \log x \implies f'' = \frac{1}{x}$, i.e., $f(x) = x \log x$ is convex if $x > 0$.

Both are widely used in statistics and data mining as loss functions,  
$\implies$ computationally tractable algorithms: least square, logistic regression.
Left panel: \( f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \) is -log convex, \( \frac{\partial^2[-\log f(x)]}{\partial x^2} = 1 > 0 \).

Right panel: a mixture of normals is not -log convex \( f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} + \frac{1}{\sqrt{2\pi10}} e^{-\frac{(x-10)^2}{200}} \)

The mixture of normals is an extremely useful model in statistics.

In general, only a local minimum can be obtained.
Procedure:

Start with an initial guess $x_0$.

Compute $x_1 = x_0 - \Delta f'(x_0)$, where $\Delta$ is the step size.

Continue the process $x_{t+1} = x_t - \Delta f'(x_t)$.

Until some criterion is met, e.g., $f(x_{t+1}) \approx f(x_t)$

The meaning of “steepest” is more clear in the two-dimensional situation.
An Example of Steepest Descent: \( f(x) = x^2 \)

\[ f(x) = x^2. \] The minimum is attained at \( x = 0, \) \( f'(x) = 2x. \)

The steepest descent iteration formula \( x_{t+1} = x_t - \Delta f'(x_t) = x_t - 2\Delta x_t. \)

Choosing the step size \( \Delta \) is important (even when \( f(x) \) is convex).

Too small \( \Delta \) \( \implies \) slow convergence, i.e., many iterations,

Too large \( \Delta \) \( \implies \) oscillations, i.e., also many iterations.
Steepest Descent in Practice

Steepest descent is one of the most widely techniques in real world

- It is extremely simple
- It only requires knowing the first derivative
- It is numerically stable (for above reasons)
- For real applications, it is often affordable to use very small $\Delta$
- In machine learning, $\Delta$ is often called learning rate
- It is used in Neural Nets and Gradient Boosting (MART)
Newton’s Method

Recall the goal is to find the $x^*$ to minimize $f(x)$.
If $f(x)$ is convex, it is equivalent to finding the $x^*$ such that $f'(x^*) = 0$.

Let $h(x) = f'(x)$. Take Taylor expansion about the optimum solution $x^*$:

$$h(x^*) = h(x) + (x^* - x)h'(x) + \text{“negligible” higher order terms}$$

Because $h(x^*) = f'(x^*) = 0$, we know approximately

$$0 \approx h(x) + (x^* - x)h'(x) \implies x^* \approx x - \frac{h(x)}{h'(x)} = x - \frac{f'(x)}{f''(x)}$$
The procedure of Newton's Method

Start with an initial guess $x_0$

Update $x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}$

Repeat $x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}$

Until some stopping criterion is reached, e.g., $x_{t+1} \approx x_t$.

An example: $f(x) = (x - c)^2$. $f'(x) = 2(x - c)$, $f''(x) = 2$.

$$x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)} \implies x_1 = x_0 - \frac{2(x_0 - c)}{2} = c$$

But we already know that $x = c$ minimizes $f(x) = (x - c)^2$.

Newton's method may find the minimum solution using only one step.
An Example of Newton’s Method: \( f(x) = x \log x \)

\[
f'(x) = \log x + 1, \quad f''(x) = \frac{1}{x}. \quad x_{t+1} = x_t - \frac{\log x_t + 1}{1/x_t}
\]

When \( x_0 \) is close to optimum solution, the convergence is very fast
When \( x_0 \) is far from the optimum, the convergence is slow initially
When \( x_0 \) is badly chosen, no convergence. This example requires \( 0 < x_0 < 1 \).
Steepest Descent for \( f(x) = x \log x \)

\[
f'(x) = \log x + 1, \quad x_{t+1} = x_t - \Delta (\log x_t + 1)
\]

Regardless of \( x_0 \), convergence is guaranteed if \( f(x) \) is convex.
May be oscillating if step size \( \Delta \) is too large
Convergence is slow near the optimum solution.
To assess the quality of the estimator $\hat{\theta}$ of $\theta$, it is common to use bias, variance, and MSE (mean square error):

\begin{align*}
\text{Bias} : & \quad E(\hat{\theta}) - \theta \\
\text{Var} : & \quad E\left(\hat{\theta} - E(\hat{\theta})\right)^2 = E(\hat{\theta}^2) - E^2(\hat{\theta}) \\
\text{MSE} : & \quad E\left(\hat{\theta} - \theta\right)^2 = \text{Var} + \text{Bias}^2
\end{align*}

The last equality is known as the bias variance trade-off. For unbiased estimators, it is desirable to have smaller variance as possible. As the sample size increases, the MLE (under certain conditions) becomes unbiased and achieves the smallest variance. Therefore, the MLE is often a desirable estimator. However, in some cases, biased estimators may achieve smaller MSE than the MLE.
The Expectations and Variances of Common Distributions

The derivations of variances are not required in this course. Nevertheless, it is useful to know the expectations and variances of common distributions.

- **Binomial**: $X \sim \text{binomial}(n, p)$, $E(X) = np$, $Var(X) = np(1 - p)$.
- **Normal**: $X \sim N(\mu, \sigma^2)$, $E(X) = \mu$, $Var(X) = \sigma^2$.
- **Chi-square**: $X \sim \chi^2(k)$, $E(X) = k$, $Var(X) = 2k$.
- **Exponential**: $X \sim \text{exp}(\lambda)$, $E(X) = \frac{1}{\lambda}$, $Var(X) = \frac{1}{\lambda^2}$.
- **Poisson**: $X \sim \text{Pois}(\lambda)$, $E(X) = \lambda$, $Var(\lambda)$. 
Multinomial Distribution

The multinomial is a natural extension to the binomial distribution.

Consider \( c \) cells and denote the observations by \((n_1, n_2, \ldots, n_c)\), which follow a \( c \)-cell multinomial distribution with the underlying probabilities \((\pi_1, \pi_2, \ldots, \pi_c)\) (with \(\sum_{i=1}^{c} \pi_i = 1\)). Denote \( n = \sum_{i=1}^{c} n_i \). We write

\[
(n_1, n_2, \ldots, n_c) \sim \text{Multinomial} (n, \pi_1, \pi_2, \ldots, \pi_c)
\]

The expectations are (for \( i = 1 \) to \( c \) and \( i \neq j \))

\[
E (n_i) = n\pi_i, \quad Var (n_i) = n\pi_i (1 - \pi_i), \quad Cov (n_i n_j) = -n\pi_i \pi_j.
\]

Note that the cells are negatively correlated (why?).
Logistic regression is one of the most widely used statistical tools for predicting categorical outcomes.

**General setup for binary logistic regression**

$n$ observations: $\{x_i, y_i\}, i = 1$ to $n$. $x_i$ can be a vector.

$y_i \in \{0, 1\}$. For example, “1” = “YES” and “0” = “NO”.

Define

$$p(x_i) = \Pr (y_i = 1|x_i) = \pi(x_i)$$

i.e., \( \Pr (y_i = 0|x_i) = 1 - p(x_i) \).
The major assumption of logistic regression

\[
\log \frac{p(x_i)}{1 - p(x_i)} = \beta_0 + \beta_1 x_{i,1} + \ldots + \beta_p x_{i,p} = \sum_{j=0}^{p} \beta_j x_{i,j}.
\]

Here, we treat \( x_{i,0} = 1 \). We can also use vector notation to write

\[
\log \frac{p(x_i; \beta)}{1 - p(x_i; \beta)} = x_i\beta.
\]

Here, we view \( x_i \) as a row-vector and \( \beta \) as a column-vector.
The model in vector notation

\[ p(x_i; \beta) = \frac{e^{x_i \beta}}{1 + e^{x_i \beta}}, \quad 1 - p(x_i; \beta) = \frac{1}{1 + e^{x_i \beta}}, \]

Log likelihood for the \(i\)th observation:

\[ l_i(\beta|x_i) = (1 - y_i) \log [1 - p(x_i; \beta)] + y_i \log p(x_i; \beta) \]

\[ = \begin{cases} 
\log p(x_i; \beta) & \text{if } y_i = 1 \\
\log [1 - p(x_i; \beta)] & \text{if } y_i = 0 
\end{cases} \]

To understand this, consider binomial with only one sample \(binomial(1, p(x_i))\) (i.e., Bernouli). When \(y_i = 1\), the log likelihood is \(\log p(x_i)\) and when \(y_i = 0\), the log likelihood is \(\log (1 - p(x_i))\). These two formulas can be written into one.
Joint log likelihood for $n$ observations:

\[
l(\beta|x_1, \ldots, x_n) = \sum_{i=1}^{n} l_i(\beta|x_i)
\]

\[
= \sum_{i=1}^{n} (1 - y_i) \log \left[ 1 - p(x_i; \beta) \right] + y_i \log p(x_i; \beta)
\]

\[
= \sum_{i=1}^{n} y_i \log \frac{p(x_i; \beta)}{1 - p(x_i; \beta)} + \log \left[ 1 - p(x_i; \beta) \right]
\]

\[
= \sum_{i=1}^{n} y_i x_i \beta - \log \left( 1 + e^{x_i \beta} \right)
\]

The remaining task is to solve the optimization problem by MLE.
Logistic Regression with Only One Variable

Basic assumption

\[ \text{logit}(\pi(x_i)) = \log \frac{p(x_i; \beta)}{1 - p(x_i; \beta)} = \beta_0 + \beta_1 x_i \]

Joint Log likelihood

\[ l(\beta | x_1, ..., x_n) = \sum_{i=1}^{n} \left[ y_i (\beta_0 + x_i \beta_1) - \log \left( 1 + e^{\beta_0 + x_i \beta_1} \right) \right] \]

Next, we solve the optimization problem for maximizing the joint likelihood, given the data.
First derivatives

\[
\frac{\partial l(\beta)}{\partial \beta_0} = \sum_{i=1}^{n} y_i - p(x_i), \quad \frac{\partial l(\beta)}{\partial \beta_1} = \sum_{i=1}^{n} x_i (y_i - p(x_i)),
\]

Second derivatives

\[
\frac{\partial^2 l(\beta)}{\partial \beta_0^2} = -\sum_{i=1}^{n} p(x_i) (1 - p(x_i)),
\]

\[
\frac{\partial^2 l(\beta)}{\partial \beta_1^2} = -\sum_{i=1}^{n} x_i^2 p(x_i) (1 - p(x_i)),
\]

\[
\frac{\partial^2 l(\beta)}{\partial \beta_0 \beta_1} = -\sum_{i=1}^{n} x_i p(x_i) (1 - p(x_i))
\]

Solve the MLE by Newton’s Method or steepest descent (two-dim problem).
Logistic Regression without Intercept ($\beta_0 = 0$)

The simplified model

$$\logit(\pi(x_i)) = \log \frac{p(x_i)}{1 - p(x_i)} = \beta x_i$$

Equivalently,

$$p(x_i) = \frac{e^{\beta x_i}}{1 + e^{\beta x_i}} = \pi(x_i), \quad 1 - p(x_i) = \frac{1}{1 + e^{\beta x_i}},$$

Joint log likelihood for $n$ observations:

$$l(\beta | x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i y_i \beta - \log \left(1 + e^{\beta x_i}\right)$$
First derivative

\[ l' (\beta) = \sum_{i=1}^{n} x_i (y_i - p(x_i)), \]

Second derivative

\[ l'' (\beta) = - \sum_{i=1}^{n} x_i^2 p(x_i) (1 - p(x_i)), \]

Newton’s Method updating formula

\[ \beta_{t+1} = \beta_t - \frac{l'(\beta_t)}{l''(\beta_t)} \]

Steepest descent (in fact ascent) updating formula

\[ \beta_{t+1} = \beta_t + \Delta l'(\beta_t) \]
A Numerical Example of Logistic Regression

Data

\[ x = \{8, 14, -7, 6, 5, 6, -5, 1, 0, -17\} \]
\[ y = \{1, 1, 0, 0, 1, 0, 1, 0, 0, 0\} \]

Log likelihood function

\[ l(\beta) \]

![Graph of log likelihood function](image)
Steepest descent is quite sensitive to the step size $\Delta$.

Too large $\Delta$ leads to oscillation.

Here, $\beta_0$ means the initial value of $\beta$. 
Newton’s Method is sensitive to the starting point $\beta_0$. May not converge at all.

The starting point (mostly) only affects computing time of steepest descent.

In general, with multiple variables, we need to use the matrix formulation, which in fact is easier to implement in matlab.
Newton’s Method for Logistic Regression with \( \beta_0 \) and \( \beta_1 \)

Analogous to the one variable case, the Newton’s update formula is

\[
\beta^{\text{new}} = \beta^{\text{old}} - \left[ \left( \frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^\top} \right)^{-1} \frac{\partial l(\beta)}{\partial \beta} \right] \beta^{\text{old}}
\]

where \( \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \),

\[
\frac{\partial l(\beta)}{\partial \beta} = \begin{bmatrix} \sum_{i=1}^{n} y_i - p(x_i) \\ \sum_{i=1}^{n} x_i (y_i - p(x_i)) \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}^\top \begin{bmatrix} y_1 - p(x_1) \\ y_2 - p(x_2) \\ \vdots \\ y_n - p(x_n) \end{bmatrix} = X^\top (y - p)
\]
\[
\left( \frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} \right) = \\
= \begin{bmatrix}
- \sum_{i=1}^{n} p(x_i)(1 - p(x_i)) & - \sum_{i=1}^{n} x_i p(x_i)(1 - p(x_i)) \\
- \sum_{i=1}^{n} x_i p(x_i)(1 - p(x_i)) & - \sum_{i=1}^{n} x_i^2 p(x_i)(1 - p(x_i))
\end{bmatrix}
= - \mathbf{X}^T \mathbf{WX}
\]

\[
\mathbf{W} = \\
= \begin{bmatrix}
p(x_1)(1 - p(x_1)) & 0 & 0 & \ldots & 0 \\
0 & p(x_2)(1 - p(x_2)) & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \ldots & p(x_n)(1 - p(x_n))
\end{bmatrix}
\]
Multivariate Logistic Regression Solution in Matrix Form

Newton’s update formula

\[ \beta^{new} = \beta^{old} - \left[ \left( \frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} \right)^{-1} \frac{\partial l(\beta)}{\partial \beta} \right]_{\beta^{old}} \]

where, in a matrix form

\[ \frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{n} x_i (y_i - p(x_i; \beta)) = X^T(y - p) \]

\[ \frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} = -\sum_{i=1}^{n} x_i^T x_i p(x_i; \beta) (1 - p(x_i; \beta)) = -X^T W X \]

We can write the update formula in a matrix form

\[ \beta^{new} = \left[ X^T W X \right]^{-1} X^T W z \]

\[ z = X \beta^{old} + W^{-1} (y - p) \]
\[ \mathbf{X} = \begin{bmatrix}
1 & x_{1,1} & x_{1,2} & \ldots & x_{1,p} \\
1 & x_{2,1} & x_{2,2} & \ldots & x_{2,p} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_{n,1} & x_{n,2} & \ldots & x_{n,p}
\end{bmatrix} \in \mathbb{R}^{n \times (p+1)} \]

\[ \mathbf{W} = \begin{bmatrix}
p_1(1 - p_1) & 0 & 0 & \ldots & 0 \\
0 & p_2(1 - p_2) & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \ldots & p_n(1 - p_n)
\end{bmatrix} \in \mathbb{R}^{n \times n} \]

where \( p_i = p(x_i; \beta^{old}) \).
Derivation

\[
\beta^{\text{new}} = \beta^{\text{old}} - \left[ \left( \frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} \right)^{-1} \frac{\partial l(\beta)}{\partial \beta} \right]_{\beta^{\text{old}}}
\]

\[
= \beta^{\text{old}} + \left[ X^T W X \right]^{-1} X^T (y - p)
\]

\[
= \left[ X^T W X \right]^{-1} \left[ X^T W X \right] \beta^{\text{old}} + \left[ X^T W X \right]^{-1} X^T (y - p)
\]

\[
= \left[ X^T W X \right]^{-1} X^T W \left( X \beta^{\text{old}} + W^{-1} (y - p) \right)
\]

\[
= \left[ X^T W X \right]^{-1} X^T W z
\]

Note that \( \left[ X^T W X \right]^{-1} X^T W z \) looks a lot like (weighted) least square.

Two major practical issues:

- The inverse may not (usually does not) exist, especially with large datasets.
- Newton update steps may be too agressive and lead to divergence.
At time $t$, update each coefficient vector:

$$\beta^t = \beta^{(t-1)} + \nu \left[ X^T W X \right]^{-1} X^T (y - p) \bigg|_{t-1}$$

where

$$W = \text{diag} \left[ p_i (1 - p_i) \right]_{i=1}^{n}$$

The magic parameter $\nu$ can be viewed as the learning rate to help make sure that the procedure converges. Practically, it is often set to be $\nu = 0.1$. 
Revisit The Simple Example with Only One $\beta$

**Data**

$x = \{8, 14, -7, 6, 5, 6, -5, 1, 0, -17\}$

$y = \{1, 1, 0, 0, 1, 0, 1, 0, 0, 0\}$

**Log likelihood function**
Newton’s Method with Learning Rate $\nu = 1$

When initial $\beta_0 = 0.32$, the method converges. When $\beta_0 = 0.33$, it does not converge.
Newton’s Method with Learning Rate $\nu = 0.1$

![Graph showing the behavior of $\beta_t$ over iterations with different initial values $\beta_0$.]
Fitting Logistic Regression With Regularization

The almost correct update formula:

$$\beta^t = \beta^{(t-1)} + \nu \left[ X^T W X + \lambda I \right]^{-1} X^T (y - p) \bigg|_{t-1}$$

Adding the regularization parameter $\lambda$ usually improves the numerical stability and sometimes may even result in better test errors.

There are also good statistical interpretations.
Fitting Logistic Regression With Regularization

The update formula:

\[ \beta^t = \beta^{(t-1)} + \nu \left[ X^T WX + \lambda I \right]^{-1} \left[ X^T (y - p) - \lambda \beta \right] \bigg|_{t-1} \]

To understand the formula, consider the following modified (regularized) likelihood function:

\[ l(\beta) = \sum_{i=1}^{n} \left\{ y_i \log p_i + (1 - y_i) \log(1 - p_i) \right\} - \frac{\lambda}{2} \sum_{j=0}^{p} \beta_j^2 \]
Newton’s Method with No Regularization $\lambda = 0$ ($\nu = 0.1$)
Newton’s Method with Regularization $\lambda = 1$ ($\nu = 0.1$)
Newton’s Method with Regularization $\lambda = 1 \ (\nu = 0.1)$
Multi-Class Logistic Regression

Data: \( \{x_i, y_i\}_{i=1}^n, x_i \in \mathbb{R}^{n \times p}, y_i \in \{0, 1, 2, \ldots, K - 1\} \).

Probability model

\[
p_{i,k} = \Pr\{y_i = k|x_i\}, \quad k = 0, 1, \ldots, K - 1,
\]

\[
\sum_{k=0}^{K-1} p_k = 1, \quad \text{(only } K - 1 \text{ degrees of freedom)}.\]

Label assignment

\[
\hat{y}_i|x_i = \arg\max_k \hat{p}_{i,k}
\]
Multi-Class Example: USPS ZipCode Recognition

Person 1:

2 4 6 8 10 12 14 16

Person 2:

2 4 6 8 10 12 14 16

Person 3:

2 4 6 8 10 12 14 16

This task can be cast (simplified) as a $K = 10$-class classification problem.
Multinomial Logit Probability Model

\[ p_{i,k} = \frac{e^{F_{i,k}}}{\sum_{s=0}^{K-1} e^{F_{i,s}}} \]

where \( F_{i,k} = F_k(x_i) \) is the function to be learned from the data.

**Linear logistic regression:** \( F_{i,k} = F_k(x_i) = x_i \beta_k \)

Note that, \( \beta_k = [\beta_{k,0}, \beta_{k,1}, \ldots, \beta_{k,p}]^T \)
Multinomial Maximum Likelihood

**Multinomial likelihood:** Suppose $y_i = k$,

$$Lik \propto p_{i,0}^0 \times \ldots \times p_{i,k}^1 \times \ldots \times p_{i,K-1}^0 = p_{i,k}$$

**Log likelihood:**

$$l_i = \log p_{i,k}, \quad \text{if } y_i = k$$

**Total log-likelihood in a double summation form:**

$$l(\beta) = \sum_{i=1}^{n} l_i = \sum_{i=1}^{n} \left\{ \sum_{k=0}^{K-1} r_{i,k} \log p_{i,k} \right\}$$

$$r_{i,k} = \begin{cases} 1 & \text{if } y_i = k \\ 0 & \text{otherwise} \end{cases}$$
Derivatives of Multi-Class Log-likelihood

The first derivative:

\[
\frac{\partial l_i}{\partial F_{i,k}} = (r_{i,k} - p_{i,k})
\]

Proof:

\[
\frac{\partial p_{i,k}}{\partial F_{i,k}} = \frac{\left[ \sum_{s=0}^{K-1} e^{F_{i,s}} \right] e^{F_{i,k}} - e^{2F_{i,k}}}{\left[ \sum_{s=0}^{K-1} e^{F_{i,s}} \right]^2} = p_{i,k} (1 - p_{i,k})
\]

\[
\frac{\partial p_{i,k}}{\partial F_{i,t}} = \frac{-e^{F_{i,k}} e^{F_{i,t}}}{\left[ \sum_{s=0}^{K-1} e^{F_{i,s}} \right]^2} = -p_{i,k} p_{i,t}
\]
\[
\frac{\partial l_i}{\partial F_{i,k}} = \sum_{s=0}^{K-1} r_{i,s} \frac{1}{p_{i,s}} \frac{\partial p_{i,s}}{\partial F_{i,k}} = r_{i,k} \frac{1}{p_{i,k}} p_{i,k}(1-p_{i,k}) + \sum_{s \neq k} r_{i,s} \frac{1}{p_{i,s}} \frac{\partial p_{i,s}}{\partial F_{i,k}} \\
= r_{i,k}(1-p_{i,k}) - \sum_{s \neq k} r_{i,s} p_{i,k} = r_{i,k} - \sum_{s=0}^{K-1} r_{i,s} p_{i,k} = r_{i,k} - p_{i,k} \]

The second derivatives:

\[
\frac{\partial^2 l_i}{\partial F_{i,k}^2} = -p_{i,k} (1-p_{i,k}) , \\
\frac{\partial^2 l_i}{\partial F_{i,k} F_{i,s}} = -p_{i,k} p_{i,s}
\]
Multi-class logistic regression can be fairly complicated. Here, we introduce a simpler approach, which does not seem to explicitly appear in common textbooks.

Conceptually, we fit $K$ binary classification problems (one vs rest) at each iteration. That is, at each iteration, we update $\beta_k$ separately for each class. At the end of each iteration, we jointly update the probabilities $p_{i,k} = \frac{e^{x_i \beta_k}}{\sum_{s=0}^{K-1} e^{x_i \beta_s}}$. 
A Simple Implementation for Multi-Class Logistic Regression

At time $t$, update each coefficient vector:

$$\beta^t_k = \beta^{(t-1)}_k + \nu \left[ X^T W_k X \right]^{-1} X^T (r_k - p_k)\Big|_{t-1}$$

where

$$r_k = [r_{1,k}, r_{2,k}, \ldots, r_{n,k}]^T$$
$$p_k = [p_{1,k}, p_{2,k}, \ldots, p_{n,k}]^T$$
$$W_k = \text{diag} \left[ p_{i,k} (1 - p_{i,k}) \right]_{i=1}^{n}$$

Then update $p_k, W_k$ for the next iteration.

Again, the magic parameter $\nu$ can be viewed as the learning rate to help make sure that the procedure converges. Practically, it is often set to be $\nu = 0.1$. 

51
Logistic Regression With $L_2$ Regularization

Total log-likelihood in a double summation form:

$$l(\beta) = \sum_{i=1}^{n} \left\{ \sum_{k=0}^{K-1} r_{i,k} \log p_{i,k} \right\} - \frac{\lambda}{2} \sum_{k=0}^{K-1} \sum_{j=0}^{d} \beta_{k,j}^2$$

$$r_{i,k} = \begin{cases} 1 & \text{if } y_i = k \\ 0 & \text{otherwise} \end{cases}$$

Let $g(\beta) = \frac{\lambda}{2} \sum_{k=0}^{K-1} \sum_{j=0}^{d} \beta_{k,j}^2$, then

$$\frac{\partial g(\beta)}{\beta_{k,j}} = \beta_{k,j} \lambda, \quad \frac{\partial^2 g(\beta)}{\beta_{k,j}^2} = \lambda$$

At time $t$, the updating formula becomes

$$\beta_{k}^{t} = \beta_{k}^{(t-1)} + \nu \left[ X^T W_k X + \lambda I \right]^{-1} \left[ X^T (r_k - p_k) - \lambda \beta_k \right]_{t-1}$$
\( L_2 \) regularization sometimes improves the numerical stability and some times may even result in better test errors.
**Logistic Regression Results on Zip Code Data**

**Zip code data:** 7291 training examples in 256 dimensions. 2007 test examples.

With no regularization ($\lambda = 0$), numerical problems may occur.
Testing Mis-Classification Error

Mis-Classification Error (%)

Iteration

λ = 0

λ = 50
Another Example on Letter \((K = 26)\) Recognition

Letter dataset: 2000 training samples in 16 dimensions. 18000 testing samples.
\[ \lambda = 0 \]

\[ \lambda = 1 \]

\[ \lambda = 10 \]

Testing error rate

\[ \text{Iterations} \]

\[ \nu = 0.1 \]

\[ ../../\text{data/letter-test.txt} \]
\[ \lambda = 0 \quad \lambda = 1 \quad \lambda = 10 \]

\[ \text{Iterations} \]

\[ \text{Log-likelihood} \]

\[ \text{ ../../data/letter-train.txt: } \nu = 0.1 \]
Simplifying Label Assignments

Recall **label assignment** in logistic regression:

\[
\hat{y}_i | x_i = \text{argmax}_k \hat{p}_{i,k}
\]

and the **probability model** of logistic regression:

\[
p_{i,k} = \frac{e^{x_i \beta_k}}{\sum_{s=0}^{K-1} e^{x_i \beta_s}}
\]

It is equivalent to assign labels directly by

\[
\hat{y}_i | x_i = \text{argmax}_k x_i \hat{\beta}_k
\]

This raises an interesting question: maybe we don’t need a probability model for the purpose of classification? For example, a linear regression may be also good?
Both linear regression and logistic regression are examples of Generalized Linear Models (GLM).

We first review linear regression and then discuss how to use it for (multi-class) classification.
Review Linear Regression

Given data \( \{x_i, y_i\}_{i=1}^n \), where \( x_i \) is a \( p \)-dimensional vector and \( y_i \) is a scalar (not limited to be categories).

We again construct the data matrix

\[
X = \begin{bmatrix}
1 & x_{1,1} & x_{1,2} & \ldots & x_{1,p} \\
1 & x_{2,1} & x_{2,2} & \ldots & x_{2,p} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_{n,1} & x_{n,2} & \ldots & x_{n,p}
\end{bmatrix}, \quad
y = \begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_n
\end{bmatrix}
\]

The data model is

\[
y = X \times \beta
\]

\( \beta \) (a vector of length \( p + 1 \)) is obtained by minimizing the mean square errors (equivalent to maximizing the joint likelihood under the normal distribution model).
Linear Regression Estimation by Least Square

The idea is to minimize the mean square errors

\[ MSE(\beta) = \sum_{i=1}^{n} |y_i - x_i \beta|^2 = (Y - X\beta)^T (Y - X\beta) \]

We can find the optimal \( \beta \) by setting the first derivative to be zero

\[ \frac{\partial MSE(\beta)}{\beta} = X^T (Y - X\beta) = 0 \]

\[ \Rightarrow X^T Y = X^T X \beta \]

\[ \Rightarrow \beta = (X^T X)^{-1} X^T Y \]

Don’t worry much about how to do matrix derivatives. The trick is to view this simply as a scalar derivative but we need to manipulate the order (and add transposes) to get the dimensions correct.
Ridge Regression

Similar to $l_2$-regularized logistic regression, we can add a regularization parameter

$$\beta = (X^TX + \lambda I)^{-1}X^TY$$

which is known as ridge regression.

Adding regularization not only improves the numerical stability but also often increases the test accuracy.
For binary classification, i.e., \( y_i \in \{0, 1\} \), we can simply treat \( y_i \) as numerical response and fit a linear regression. To obtain the classification result, we can simply use \( \hat{y} = 0.5 \) as the classification threshold.

Multi-class classification (with \( K \) classes) is more interesting. We can use exactly the same trick as in multi-class logistic regression by first expanding the \( y_i \) into a vector of length \( K \) with only one entry being 1 and then fitting \( K \) binary linear regressions simultaneously and using the location of the maximum fitted value as the class label prediction.
This is essentially the first iteration of multi-class logistic regression. Clearly, the results are not as good as logistic regression with many iterations.

Adding regularization ($\lambda$) slightly increases the training errors but decreases the testing errors at certain range.