LEC 5: Logistic Regression

Guangliang Chen

October 22, 2018

Outline

- Introduction
- Logistic regression (2 classes)
- Softmax regression (3 classes or more)
- Summary

Classification is a special instance of regression

Classification is essentially a regression problem with discrete outputs (i.e., a small, finite set of values):

$$\underbrace{y}_{\text{discrete response}} = f\,(\ \underbrace{x_1, \ldots, x_d}_{\text{features (predictors)}}\ ).$$

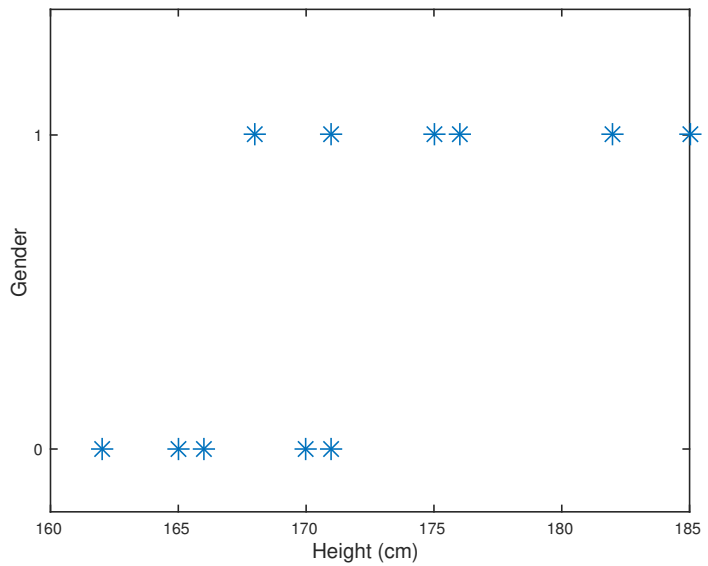
Thus, it can be approached from a regression point of view.

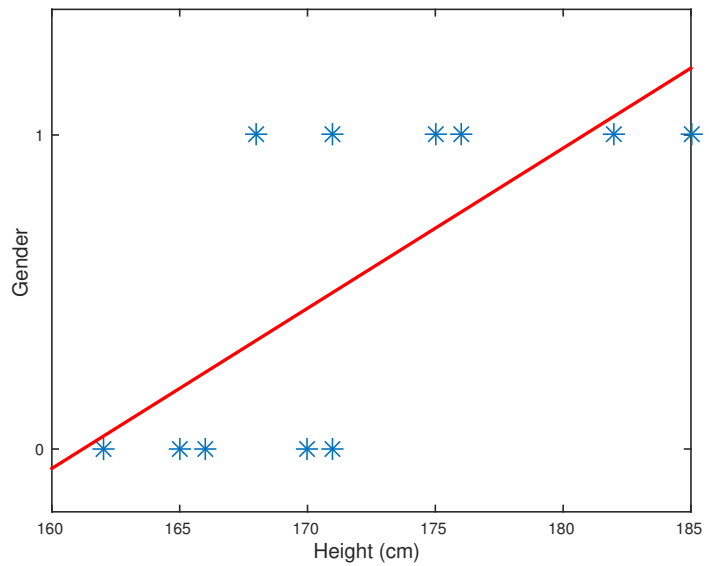
To explain ideas, we start with the 1-D binary classification problem which has only one predictor x and a binary response $y = 0$ or 1:

$$\underbrace{y}_{\text{binary response}} = f(\ \underbrace{x}_{\text{predictor}}\ ).$$

Motivating example

Consider a specific example where x represents a person's height while y denotes the person's sex (0 = Female, 1 = Male).

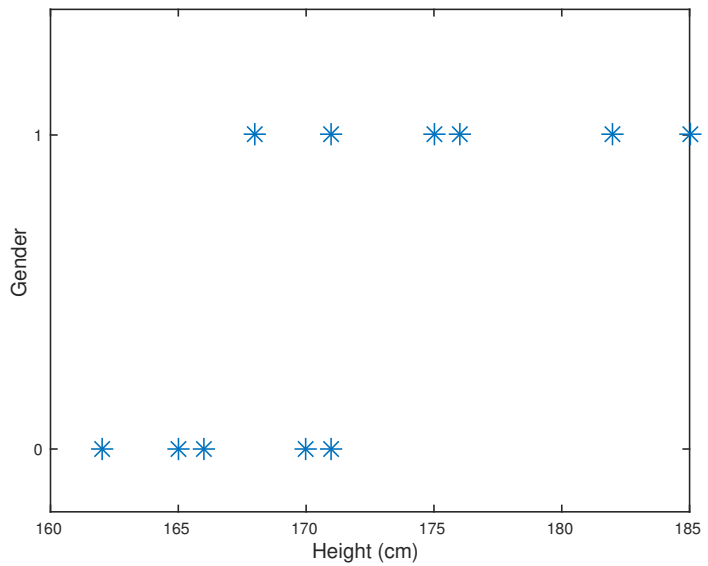


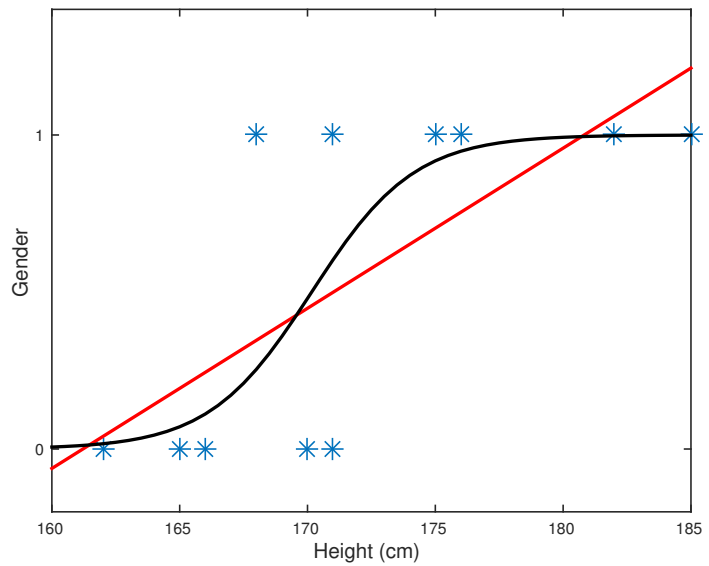


Simple linear regression is obviously not appropriate in this case.

Motivating example

Consider a specific example where x represents a person's height while y denotes the person's sex (0 = Female, 1 = Male).



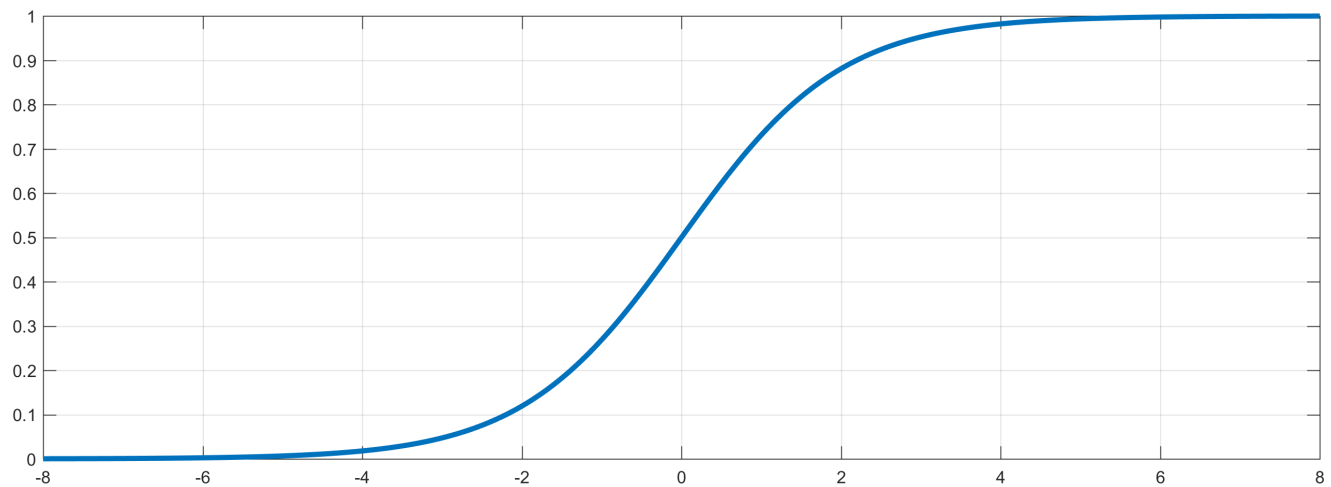


A better option is to use an S-shaped curve to fit the data.

Which functions have such shapes?

An example is the **logistic/sigmoid** function:

$$g(z) = \frac{1}{1+e^{-z}}, \quad -\infty < z < \infty$$



Can you think of another function that has such a shape?

Properties of the logistic function

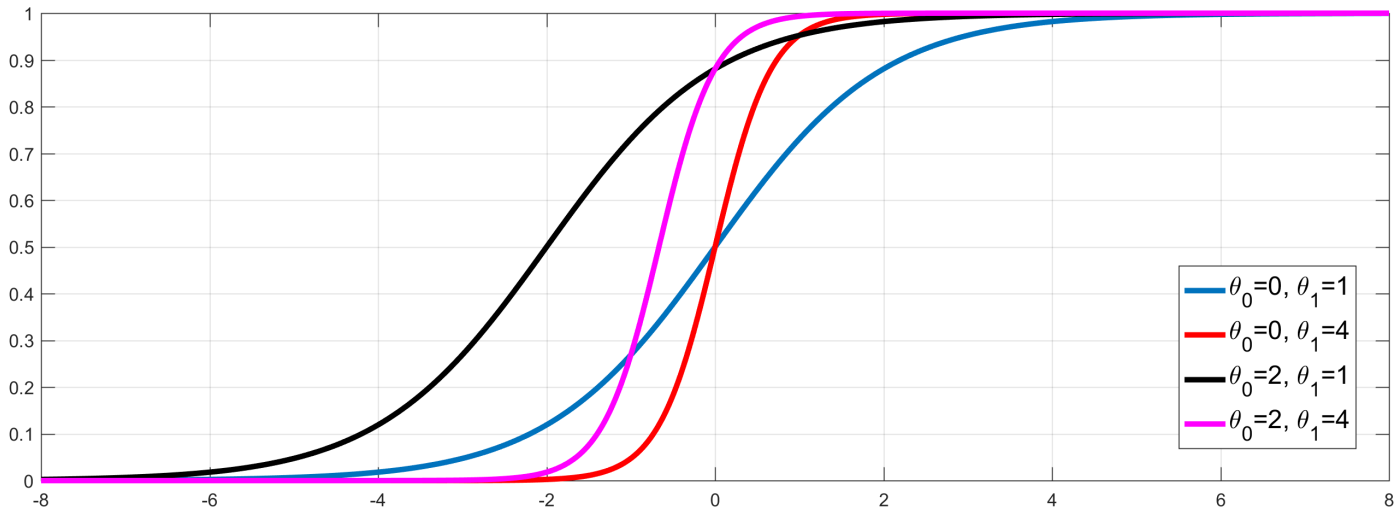
- $g(z)$ is defined for all real numbers z
- $g(z)$ is monotonically increasing over its domain
- $0 < g(z) < 1$ for all $z \in \mathbb{R}$
- $g(0) = 0.5$
- $\lim_{z \to -\infty} g(z) = 0$ and $\lim_{z \to +\infty} g(z) = 1$
- $g'(z) = g(z)(1 - g(z))$ for any z. $\longleftarrow$ This is a very important property

Making the logistic function more flexible

We generalize the logistic function to a **location-scale family**:

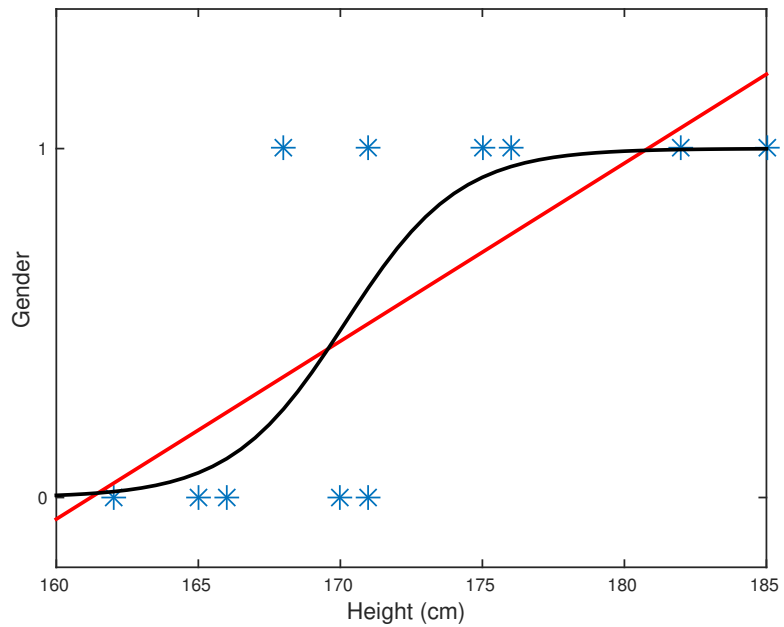
$$g(\theta_0 + \theta_1 x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$

where $\theta_0 \in \mathbb{R}$ is a location parameter and $\theta_1 > 0$ is a scale parameter.



The logistic regression problem

Once we fix the template function $g(z)$, the logistic regression problem reduces to parameter estimation based on a set of examples.



Problem.
Given training data $(x_i, y_i), 1 \leq i \leq n$, find θ_0, θ_1 such that the curve

$$y = g(\theta_0 + \theta_1 x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$

fits the data in some optimal way.

How to define the optimality

There are two different ways:

- **Optimization** approach.
- **Statistical** approach.

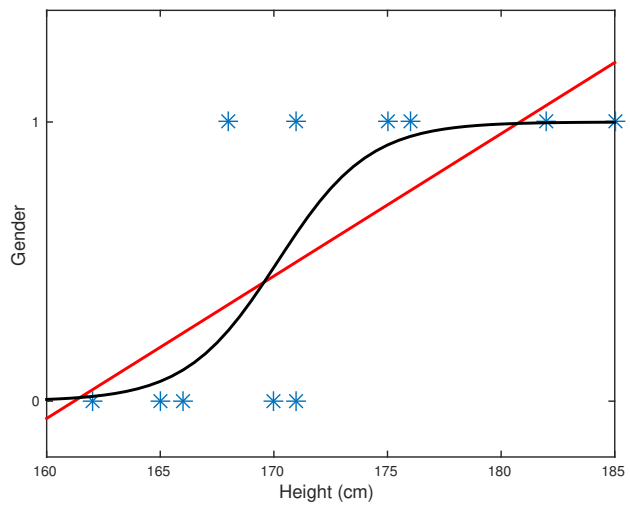
The optimization approach

We regard $\hat{y} = g(\theta_0 + \theta_1 x)$ as the fitted value at x and use a *loss* function, e.g., square loss $\ell(y, \hat{y}) = (y - \hat{y})^2$, to quantify the goodness of fit of any such curve:

$$\min_{\boldsymbol{\theta}=(\theta_0,\theta_1)} \sum_{i=1}^{n} \ell(y_i, \hat{y}_i), \qquad \text{where} \quad \hat{y}_i = g(\theta_0 + \theta_1 x_i)$$

In the above,

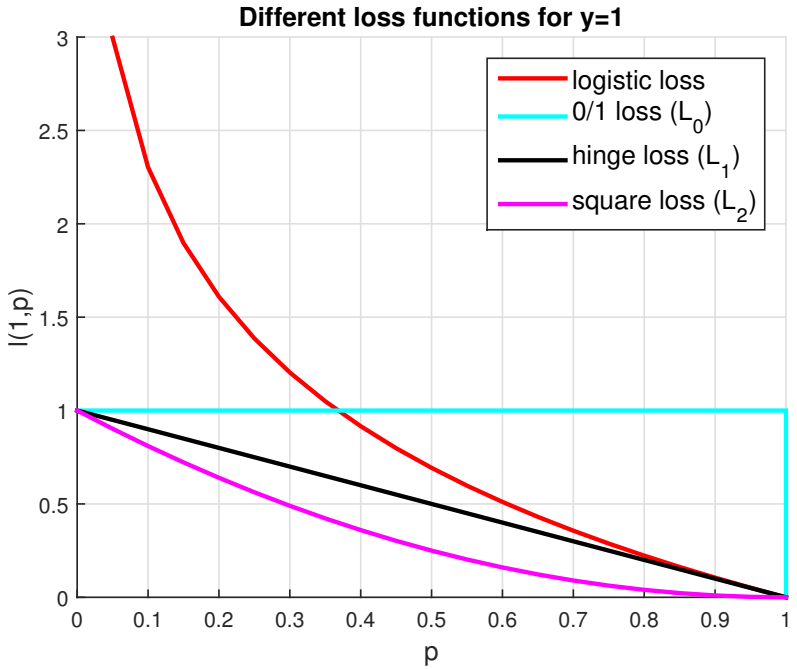
- y_i: the ith observation
- $\hat{y}_i$: the fitted value at x_i
- $\sum \ell(y_i, \hat{y}_i)$: total loss



Clearly, the previous formulation is independent of the choices of the template function g and of the loss function ℓ.

Commonly-used loss functions:

- $0/1$ loss: $\ell(y, \hat{y}) = 1_{y \neq \hat{y}}$
- Square loss: $\ell(y, \hat{y}) = (y - \hat{y})^2$
- Hinge loss: $\ell(y, \hat{y}) = |y - \hat{y}|$
- Logistic loss: $\ell(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$



For example, when the square loss is used (along with the sigmoid template function), the problem becomes the following:

$$\min_{\boldsymbol{\theta}=(\theta_0,\theta_1)} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

where

$$\hat{y}_i = g(\theta_0 + \theta_1 x_i) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_i)}}$$

Computing tool: multivariable calculus

The gradient of the total loss (as a function of θ_0, θ_1) is

$$\frac{\partial}{\partial \boldsymbol{\theta}} \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) = \left(\sum_{i=1}^{n} \frac{\partial \ell}{\partial \theta_0}, \sum_{i=1}^{n} \frac{\partial \ell}{\partial \theta_1} \right) \Bigg|_{(y_i, \hat{y}_i)}$$

One approach to minimizing the total loss is to find its critical points

$$\sum_{i=1}^{n} \frac{\partial \ell}{\partial \theta_0} = 0, \quad \sum_{i=1}^{n} \frac{\partial \ell}{\partial \theta_1} = 0$$

However, this often leads to very complex equations.

For example, when the square loss is used, the gradient can be shown to be

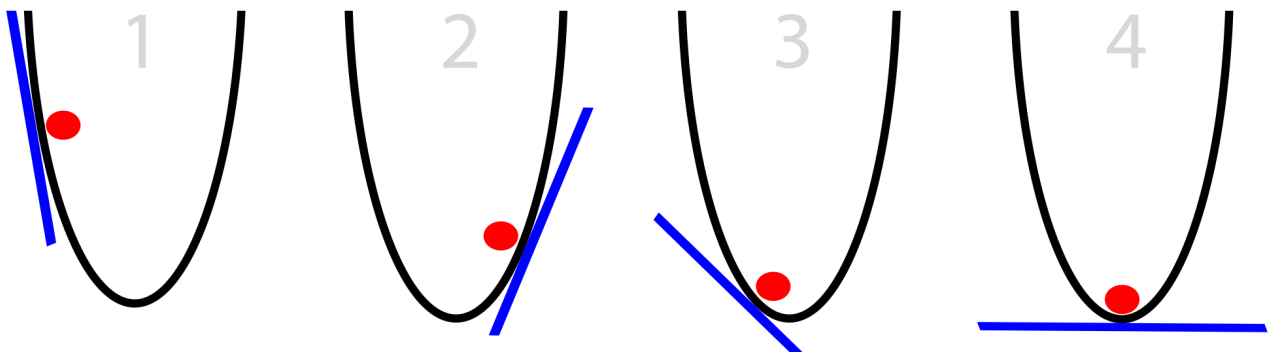
$$-2\left(\sum_{i=1}^{n}\hat{y}_i(1-\hat{y}_i)(y_i-\hat{y}_i),\ \sum_{i=1}^{n}x_i\hat{y}_i(1-\hat{y}_i)(y_i-\hat{y}_i)\right)$$

Proof:

Finding the critical points in this case is highly nontrivial and in practice, one often resorts to **Newton's method**[1] for finding the roots.

[1] `http://tutorial.math.lamar.edu/Classes/CalcI/NewtonsMethod.aspx`

A second approach to finding the minimum is through **gradient descent**: Start from some location and always move by a small amount in the direction of largest decrease (i.e., negative gradient):



If the function is $f(x)$, the current location is $x^{(t)}$, then the next move will be

$$x^{(t+1)} = x^{(t)} - \eta \cdot f'(x^{(t)})$$

where $\eta > 0$ is a parameter, called learning rate.

This approach can also be generalized to higher dimensions:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t+1)} - \eta \cdot \nabla f(\mathbf{x}^{(t)}), \quad \mathbf{x}^{(0)} \text{ specified by user}$$

This is one of the most important and popular techniques in machine learning.

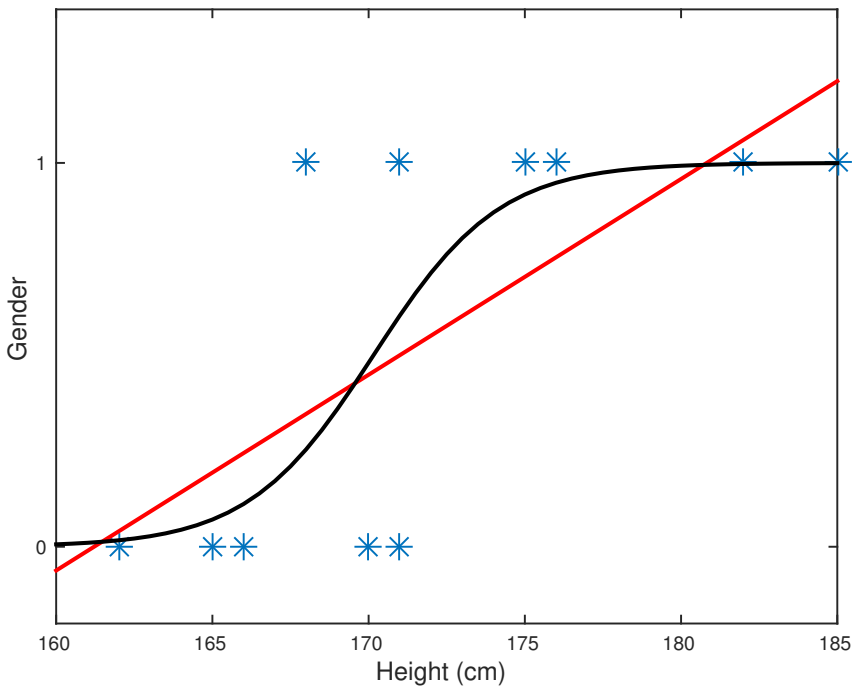
In our setting, the gradient descent rule is the following

$$\theta_0^{(t+1)} := \theta_0^{(t)} - \eta \sum_{i=1}^{n} \frac{\partial \ell}{\partial \theta_0}(y_i, \hat{y}_i)\Big|_{\theta_0^{(t)}, \theta_1^{(t)}}$$

$$\theta_1^{(t+1)} := \theta_1^{(t)} - \eta \sum_{i=1}^{n} \frac{\partial \ell}{\partial \theta_1}(y_i, \hat{y}_i)\Big|_{\theta_0^{(t)}, \theta_1^{(t)}}$$

The statistical way

Key: We will interpret $p(x, \boldsymbol{\theta}) = g(\theta_0 + \theta_1 x) = \frac{1}{1+e^{-(\theta_0+\theta_1 x)}}$ as probability!



More specifically, assume that there is a probabilistic distribution for Y (sex) at each given x (height):

$$P(Y = 1 \mid x) = p(x; \boldsymbol{\theta}),$$
$$P(Y = 0 \mid x) = 1 - p(x; \boldsymbol{\theta})$$

This implies that

$$Y \mid x \sim \text{Bernoulli}(p = p(x; \boldsymbol{\theta}))$$

with associated pmf

$$f(y; p) = P(Y = y \mid x) = p^y (1 - p)^{1-y}, \quad \text{for } y = 0, 1$$

Assuming independent training examples $(x_i, y_i), 1 \leq i \leq n$, the likelihood function of the sample is

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} f(y_i; p(x_i; \boldsymbol{\theta})) = \prod_{i=1}^{n} p(x_i; \boldsymbol{\theta})^{y_i} (1 - p(x_i; \boldsymbol{\theta}))^{1-y_i}$$

The maximizer of $L(\boldsymbol{\theta})$, called the Maximum Likelihood Estimator (MLE) of $\boldsymbol{\theta}$, gives the optimal parameter values for the model.

This is also a very important tool for machine learning (whenever parameter estimation for a distribution is involved).

Connection to the optimization approach

In practice, the MLE of $\boldsymbol{\theta}$ is obtained by maximizing the *log* likelihood function

$$\max_{\boldsymbol{\theta}} \log L(\boldsymbol{\theta}) = \sum_{i=1}^{n} y_i \log p(x_i; \boldsymbol{\theta}) + (1 - y_i) \log(1 - p(x_i; \boldsymbol{\theta}))$$

Mathematically, it is equivalent to the following minimization problem

$$\min_{\boldsymbol{\theta}} -\log L(\boldsymbol{\theta}) = \sum_{i=1}^{n} (-y_i \log p(x_i; \boldsymbol{\theta}) - (1 - y_i) \log(1 - p(x_i; \boldsymbol{\theta})))$$

This is exactly optimization with the logistic loss

$$\ell(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

Finding the MLE of θ

This falls under the previous setting of optimization via multivariable calculus in two ways. However, in this case, things are quite simple.

First, it can be shown that the gradient of the log likelihood function is

$$\left(\frac{\partial \log L(\theta)}{\partial \theta_0}, \frac{\partial \log L(\theta)}{\partial \theta_1}\right) = \left(\sum_{i=1}^{n}(y_i - p(x_i; \boldsymbol{\theta})), \sum_{i=1}^{n}(y_i - p(x_i; \boldsymbol{\theta}))x_i\right)$$

Proof:

The same two methods can be used to find the MLE:

- **Critical-point** method:

$$0 = \sum_{i=1}^{n} (y_i - p(x_i; \boldsymbol{\theta}))$$

$$0 = \sum_{i=1}^{n} (y_i - p(x_i; \boldsymbol{\theta}))x_i$$

  Due to the complex form, *Newton's iteration* has to be used.

- **Gradient descent**: Always move by a small amount in the direction of largest increase (i.e., gradient):

$$\theta_0^{(t+1)} := \theta_0^{(t)} + \eta \cdot \sum_{i=1}^{n} (y_i - p(x_i; \boldsymbol{\theta}^{(t)}))$$

$$\theta_1^{(t+1)} := \theta_1^{(t)} + \eta \cdot \sum_{i=1}^{n} (y_i - p(x_i; \boldsymbol{\theta}^{(t)}))x_i$$

  in which $\eta > 0$ is the *learning rate*.

How to classify new observations

After we fit the logistic model to the training set,

$$p(x;\boldsymbol{\theta}) = \frac{1}{1+e^{-(\theta_0+\theta_1 x)}}$$

we may use the following decision rule for a new observation x^*:

$$\text{Assign label } y^* = 1 \quad \text{if and only if} \quad p(x^*;\boldsymbol{\theta}) > \frac{1}{2}.$$

Remark. Logistic regression is actually also a kind of Bayes classifier because it computes posterior probabilities at every x:

$$p(x;\boldsymbol{\theta}) = P(Y = 1 \mid x)$$

MATLAB functions for logistic regression

```
x = [162 165 166 170 171 168 171 175 176 182 185]';

y = [0 0 0 0 0 1 1 1 1 1 1]';

glm = fitglm(x, y, 'linear', 'distr', 'binomial');

p = predict(glm, x);

% p = [0.0168, 0.0708, 0.1114, 0.4795, 0.6026, 0.2537, 0.6026, 0.9176,
0.9483, 0.9973, 0.9994]
```

Python scripts for logistic regression

```
import numpy as np
from sklearn import linear_model

x = np.transpose(np.array([[162, 165, 166, 170, 171, 168, 171, 175, 176,
182, 185]]))

y = np.transpose(np.array([[0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]]))

logreg = linear_model.LogisticRegression(C=1e5).fit(x, y.ravel())

prob = logreg.predict_proba(x) # fitted probabilities

pred = logreg.predict(x) # prediction of labels
```

R script

```
x = c(162, 165, 166, 170, 171, 168, 171, 175, 176, 182, 185)

y = c(0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1)

model <- glm(y~x,family=binomial(link='logit'))

p = model$fitted.values
# p = [0.0168, 0.0708, 0.1114, 0.4795, 0.6026, 0.2537, 0.6026, 0.9176,
0.9483, 0.9973, 0.9994]

theta = model$coefficients

fitted.prob <- predict(model,data.frame(x=c(167,170,175)),type='response')
```

The general binary classification problem

When there are multiple predictors $x_1, \ldots, x_d$, we let

$$p(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d)}}.$$

Still the same procedure to find the best $\boldsymbol{\theta}$.

The classification rule also remains the same:

$$y = 1_{p(\mathbf{x};\boldsymbol{\theta}) > 0.5}$$

We call this classifier the Logistic Regression classifier.

Understanding LR: decision boundary

The decision boundary consists of all points $\mathbf{x} \in \mathbb{R}^d$ such that

$$p(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{2}$$

or equivalently,

$$\theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d = 0$$

This is a hyperplane showing that LR is a linear classifier.

Understand LR: model

The LR model can be rewritten as

$$\log \frac{p}{1-p} = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d = \boldsymbol{\theta} \cdot \mathbf{x}$$

where $x_0 = 1$ (for convenience) and

- p: probability of "success" (i.e. $Y = 1$)
- $\frac{p}{1-p}$: odds of "winning"
- $\log \frac{p}{1-p}$: logit (a link function)

Remark. LR belongs to a family called *generalized linear models* (GLM).

Multiclass extensions

We have introduced logistic regression in the setting of binary classification.

There are two ways to extend it for multiclass classification:

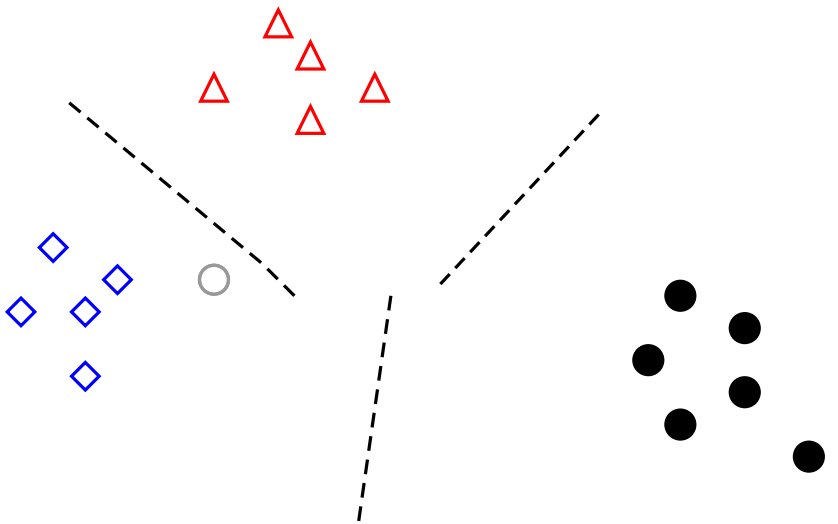
- **Union of binary models**
  - *One versus one*: construct a LR model for every pair of classes
  - *One versus rest*: construct a LR model for each class against the rest of training set

  In either case, the "most clearly winning" class is adopted as the final prediction.
- **Softmax Regression** (fixed versus rest)

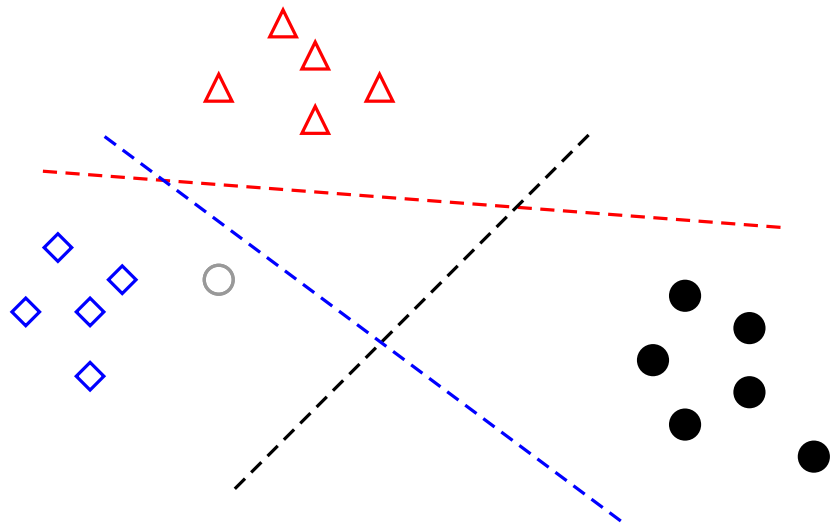
Union of binary models

Logistic regression can be extended to a multiclass setting in two ways:

One-versus-one extension



One-versus-rest extension



In either case, the "maximally winning" class is adopted as the final prediction:

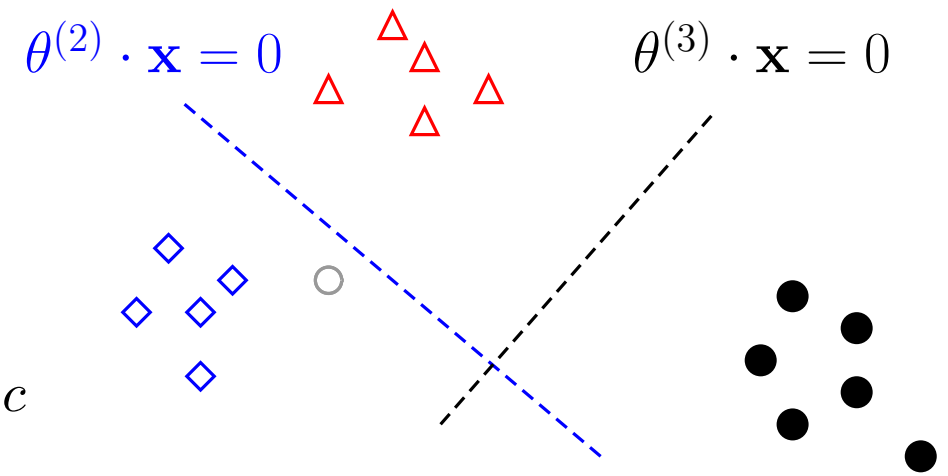
- For the one-versus-one extension, the final prediction for a test point is the most frequent label predicted;
- For the one-versus-rest extension,
  - The reference classes are assigned label 1 (the rest with label 0)
  - For each binary model, record the 'score': $p(\mathbf{x}, \boldsymbol{\theta}^{(j)})$ (not the predicted label)
  - The final prediction is the reference class with the highest score

$$y = \arg\max_j \; p(\mathbf{x}, \boldsymbol{\theta}^{(j)})$$

What is softmax regression?

Softmax regression fixes one class (say class 1) and fits $c-1$ binary logistic regression models for each of the remaining classes against that class:

$$\log \frac{P(Y=j \mid \mathbf{x})}{P(Y=1 \mid \mathbf{x})} = \boldsymbol{\theta}^{(j)} \cdot \mathbf{x}, \quad 2 \le j \le c$$



Equivalently, this is

$$P(Y=j \mid \mathbf{x}) = P(Y=1 \mid \mathbf{x}) \cdot e^{\boldsymbol{\theta}^{(j)} \cdot \mathbf{x}}$$

The prediction for a new observation is the class with max (relative) probability:

$$\hat{y} = \arg\max \{0,\ \boldsymbol{\theta}^{(2)} \cdot \mathbf{x},\ \ldots,\ \boldsymbol{\theta}^{(c)} \cdot \mathbf{x}\}$$

Solving the system together with the constraint

$$\sum_{j=1}^{c} P(Y = j \mid \mathbf{x}) = 1$$

yields that

$$P(Y = 1 \mid \mathbf{x}) = \frac{1}{1 + \sum_{j=2}^{c} e^{\boldsymbol{\theta}^{(j)} \cdot \mathbf{x}}} \quad \longleftarrow \ell = 1$$

and correspondingly,

$$P(Y = \ell \mid \mathbf{x}) = \frac{e^{\boldsymbol{\theta}^{(\ell)} \cdot \mathbf{x}}}{1 + \sum_{j=2}^{c} e^{\boldsymbol{\theta}^{(j)} \cdot \mathbf{x}}}, \quad \ell = 2, \ldots, c$$

Remark. If we define $\boldsymbol{\theta}^{(1)} = \mathbf{0}$, then the two sets of formulas may be unified

$$P(Y = \ell \mid \mathbf{x}) = \frac{e^{\boldsymbol{\theta}^{(\ell)} \cdot \mathbf{x}}}{\sum_{j=1}^{c} e^{\boldsymbol{\theta}^{(j)} \cdot \mathbf{x}}}, \quad \ell = 1, \ldots, c$$

The new formula is actually shift-invariant with respect to the parameters

$$P(Y=\ell \mid \mathbf{x}) = \frac{e^{\boldsymbol{\theta}^{(\ell)}\cdot \mathbf{x}}}{\sum_{j=1}^{c} e^{\boldsymbol{\theta}^{(j)}\cdot \mathbf{x}}} = \frac{e^{(\boldsymbol{\theta}^{(\ell)}+\boldsymbol{t})\cdot \mathbf{x}}}{\sum_{j=1}^{c} e^{(\boldsymbol{\theta}^{(j)}+\boldsymbol{t})\cdot \mathbf{x}}}, \quad \ell = 1, \ldots, c$$

We may thus relax the constant $\boldsymbol{\theta}^{(1)} = \mathbf{0}$ to be a parameter so that we have a symmetric model, with (redundant) parameters $\Theta = \{\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(c)}\}$.

Remark. The form of the above probabilities is the so-called **softmax function**:

$$\operatorname{softmax}(\ell, \alpha_1, \ldots, \alpha_c) = \frac{e^{\alpha_\ell}}{\sum_{j=1}^{c} e^{\alpha_j}}$$

It is a smooth function trying to approximate the indicator function

$$1_{\alpha_\ell \text{ is largest}} = \begin{cases} 1 & \alpha_\ell \text{ is largest} \\ 0, & \text{otherwise} \end{cases}.$$

The distribution of Y, taking discrete values $1, \ldots, c$, is multinomial with the corresponding probabilities

$$p_j = P(Y = j \mid \mathbf{x}; \Theta) = \frac{e^{\boldsymbol{\theta}^{(j)} \cdot \mathbf{x}}}{\sum_{i=1}^{c} e^{\boldsymbol{\theta}^{(i)} \cdot \mathbf{x}}}, \quad j = 1, \ldots, c$$

Therefore, softmax regression is also called **multinomial logistic regression**.

The decision rule for any new observation is

$$\hat{y} = \arg\max_j p_j = \arg\max_j P(Y = j \mid \mathbf{x}; \Theta)$$

Clearly, it is also a Bayes classifier.

Parameter estimation

Like logistic regression, softmax regression estimates the parameters by maximizing the likelihood of the training set:

$$L(\Theta) = \prod_{i=1}^{n} P(Y = y_i \mid \mathbf{x}_i; \Theta) = \prod_{i=1}^{n} \frac{e^{\boldsymbol{\theta}^{(y_i)} \cdot \mathbf{x}_i}}{\sum_{j=1}^{c} e^{\boldsymbol{\theta}^{(j)} \cdot \mathbf{x}_i}}$$

The MLE can be found by using either Newton's method or gradient descent.

MATLAB functions for multinomial LR

```
x = [162 165 166 170 171 168 171 175 176 182 185]';

y = [0 0 0 0 0 1 1 1 1 1 1]';

B = mnrfit(x,categorical(y));

p = mnrval(B, x);
```

Python function for multinomial LR

```
logreg = linear_model.LogisticRegression(C=1e5, multi_class=
'multinomial', solver='newton-cg').fit(x, y.ravel())

# multi_class = 'ovr' (one versus rest) by default

# solver='lbfgs' would also work (default ='liblinear')
```

Feature selection for logistic regression

Logistic regression tends to overfit the data in the setting of high dimensional data (i.e., many predictors). There are two ways to resolve this issue:

- First use a **dimensionality reduction** method (such as PCA, LDA) to project data into lower dimensions
- Add a **regularization** term to the objective function

$$\min_{\boldsymbol{\theta}=(\theta_0,\theta_1)} -\sum_{i=1}^{n} y_i \log p(x_i;\boldsymbol{\theta}) + (1-y_i)\log(1-p(x_i;\boldsymbol{\theta})) + C\|\boldsymbol{\theta}\|_p^p$$

where p is normally set to 2 (ℓ_2 regularization) or 1 (ℓ_1 regularization). The constant $C > 0$ is called a regularization parameter; larger values of C would lead to sparser (simpler) models.

Python function for regularized LR

with default values
logreg = linear_model.LogisticRegression(penalty='l2', C=1.0, solver='liblinear', multi_class='ovr')

penalty: may be set to 'l1'
C: inverse of regularization strength (smaller values specify stronger regularization). Cross-validation is often needed to tune this parameter.
multi_class: may be changed to 'multinomial' (no 'ovo' option)
solver: {'newton-cg', 'lbfgs', 'liblinear', 'sag'}. Algorithm to use in the optimization problem.

(to be continued)

(cont'd from last page)

solver: {'newton-cg', 'lbfgs', 'liblinear', 'sag'}. Algorithm to use in the optimization problem.

- For small datasets, 'liblinear' is a good choice, whereas 'sag' is faster for large ones.
- For multiclass problems, only 'newton-cg' and 'lbfgs' handle multinomial loss; 'sag' and 'liblinear' are limited to one-versus-rest schemes.
- 'newton-cg', 'lbfgs' and 'sag' only handle L2 penalty.

HW5 (due Monday, Nov 5)

1. Implement Newton's method to determine an approximation to the solution to $\cos x = x$ that lies in the interval $[0, \pi/2]$. Use three different initializations $x_0 = 0, 1, \pi/2$ and plot the respective sequences of numbers that converge to the exact solution. What is your conclusion?

2. Use gradient descent to find the local minimum of the following function

$$f(x) = \frac{1}{4}x^4 - 3x^3 + 7x - 5$$

around $x = 0$. Try three different learning rates $\eta = 0.01, 0.05, 0.1$ and plot the respective sequences (up to 30 iterations). Which learning rate is the best?

3. Apply the binary logistic regression classifier to the data in HW4-Q5 to compare with LDA in terms of decision boundary and test accuracy.

4. Implement the three multiclass extensions of the binary logistic regression classifier (one-vs-one, one-vs-rest, and multinomial) and apply them to the USPS data set (after PCA 95%). Compare with the multiclass LDA classifier in terms of test accuracy.

5. Repeat the previous question with the following subsets of the MNIST data set: (a) 4, 9, (b) 0, 1, 2 (c) 3, 5, 8

6 (Extra credit) Apply the ℓ_1-regularized multinomial logistic regression to the USPS data set (no pca is needed). How does it compare with those in Question 4?