LEC 6: Logistic Regression

Dr. Guangliang Chen

March 10, 2016

Outline

- Logistic regression
- Softmax regression
- Summary

Classification is a special kind of regression

Classification is essentially a regression problem with discrete outputs (i.e., a small, finite set of values). Thus, it can be approached from a regression point of view.

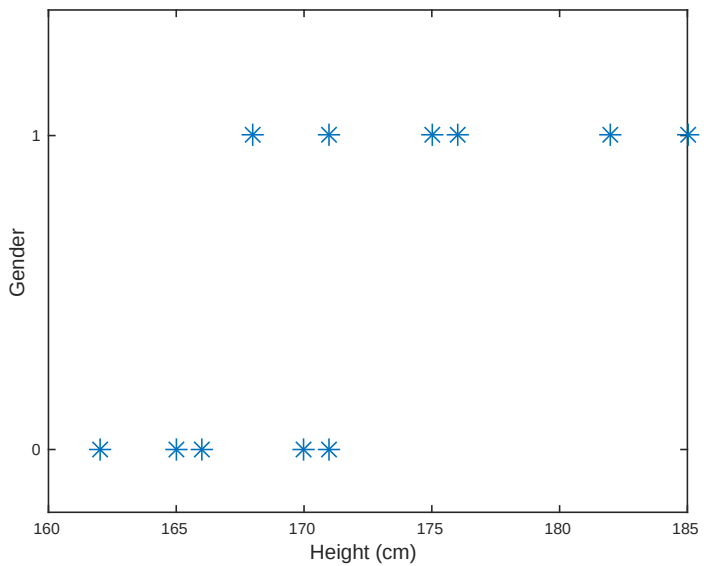
In our case, there are 784 predictors (pixels) while the target variable is categorical (with 10 possible values $0, 1, \ldots, 9$):

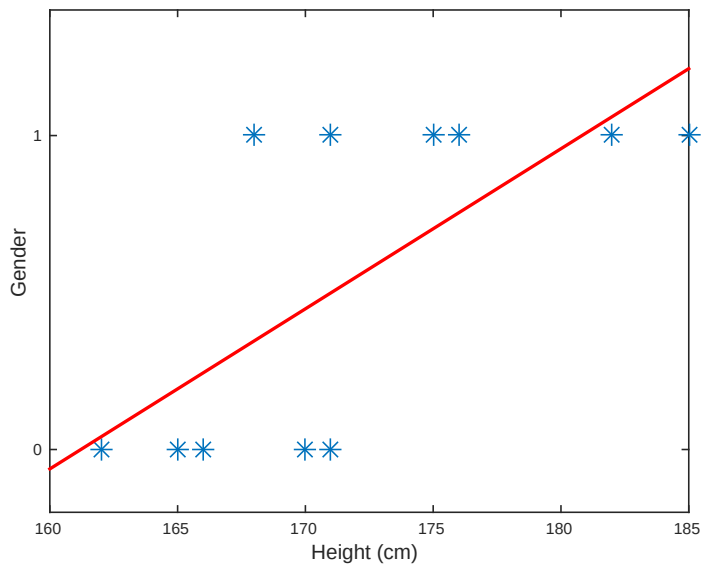
$$y \approx f(x_1, \ldots, x_{784}).$$

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

Motivating example

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

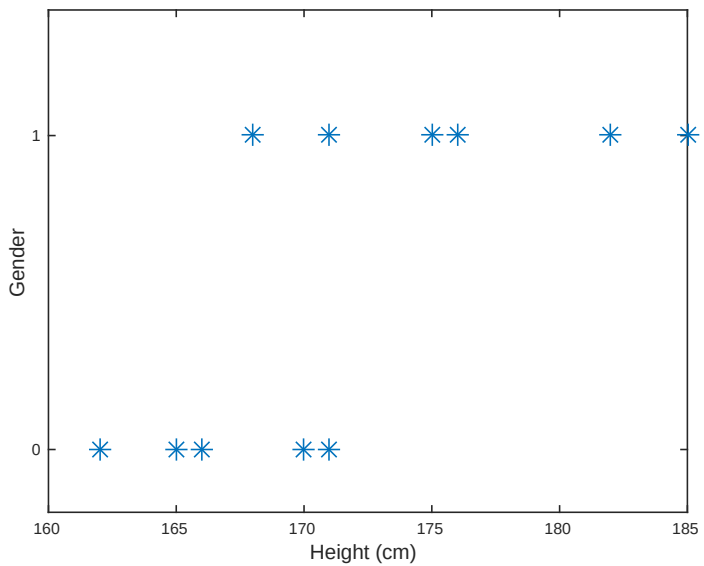


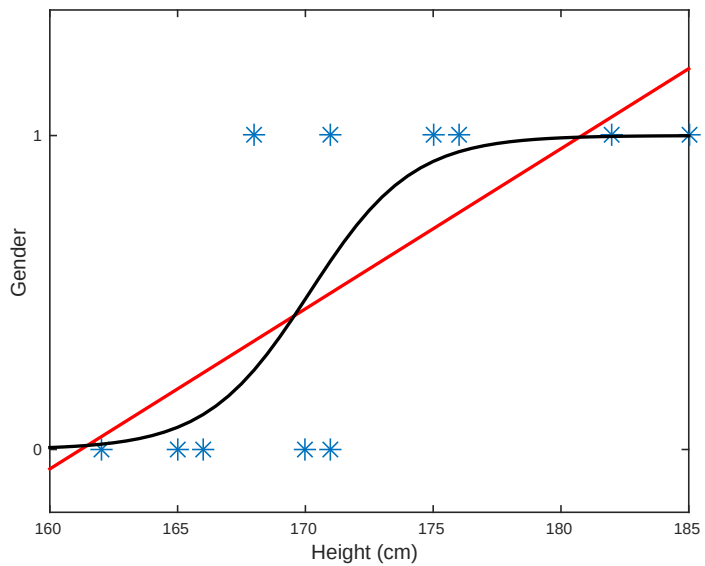


Simple linear regression is obviously not appropriate in this case.

Motivating example

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





A better choice is to use a curve that adapts to the shape.

Such a curve may be obtained by using the following family of functions

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

where

- The template is $g(z) = \frac{1}{1+e^{-z}}$, called the **logistic/sigmoid** function.
- The parameters θ_0, θ_1 control *location* and *sharpness* of jump respectively.

Properties of the logistic function

- $g(z)$ is defined for all real numbers z
- $g(z)$ is a monotonically increasing function
- $0 < g(z) < 1$ 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

How to estimate $\vec{\theta}$

- **Optimization** problem:

$$\min_{\vec{\theta}} \sum_{i=1}^{n} \ell(y_i, p(x_i; \vec{\theta}))$$

  where ℓ is a *loss* function, e.g., square loss $\ell(y, p) = (y - p)^2$.

- **Probabilistic** perspective: We regard gender (y) as a random variable (Y) and interpret $p(x; \vec{\theta})$ as probability:

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

  This implies that $\mathrm{E}(Y \mid x; \vec{\theta}) = p(x; \vec{\theta})$.
  Clearly,

$$Y \mid x; \vec{\theta} \sim \text{Bernoulli}(p(x; \vec{\theta})).$$

The pdf of $Y \sim$ Bernoulli(p) can be written as

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

Assuming that the training examples were generated independently, the likelihood function of the sample is

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

and the log likelihood is

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

Maximum Likelihood Estimation (MLE)

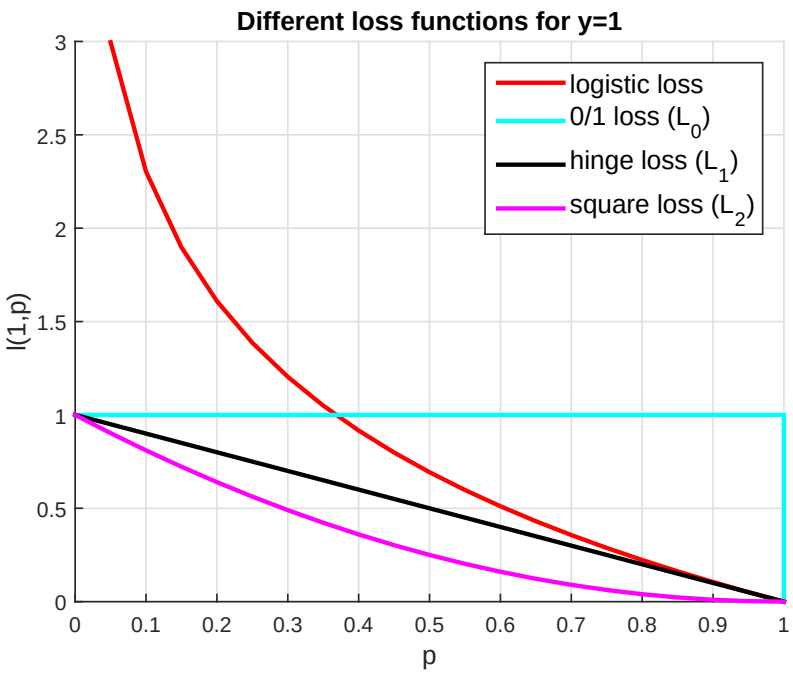
In principle, the MLE of $\vec{\theta}$ is obtained by maximizing the log likelihood function

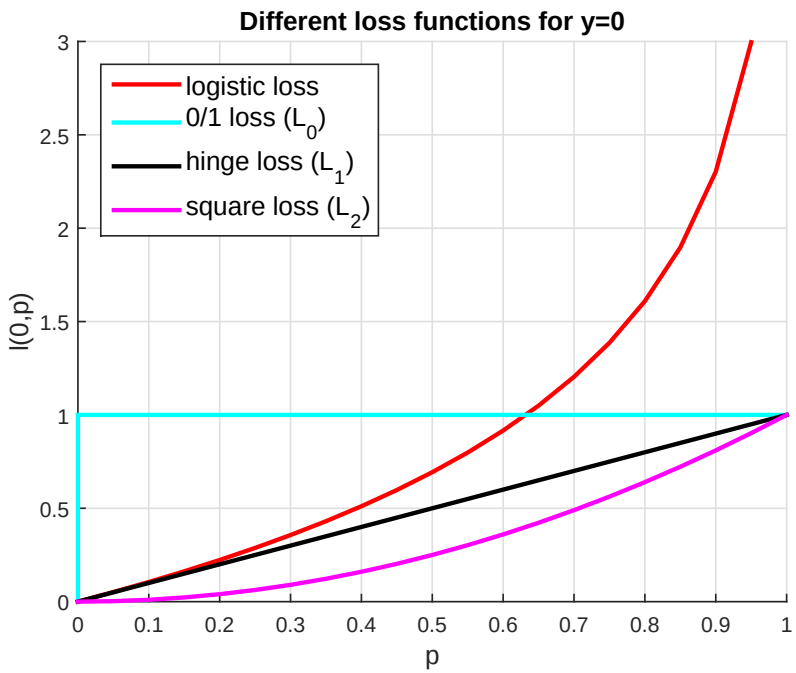
$$\begin{aligned}\log L(\vec{\theta}) &= \sum_{i=1}^{n} y_i \log p(x_i;\vec{\theta}) + (1-y_i)\log(1-p(x_i;\vec{\theta})) \\ &= \sum_{i=1}^{n} y_i \log \frac{1}{1+e^{-(\theta_0+\theta_1 x_i)}} + (1-y_i)\log(1-\frac{1}{1+e^{-(\theta_0+\theta_1 x_i)}})\end{aligned}$$

This actually corresponds to optimization with the logistic loss function

$$\ell(y,p) = -(y\log p + (1-y)\log(1-p)) = \begin{cases} -\log p, & y=1; \\ -\log(1-p), & y=0 \end{cases}$$

Loss functions





Finding the MLE of $\vec{\theta}$

It can be shown that the gradient of the log likelihood function is

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

There are two ways to find the MLE:

- **Critical-point** method:

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

Due to the complex form *Newton's iteration* is used. In one dimension, the method works as follows:

$$\text{Solve } f(\theta) = 0 \qquad \text{by update rule} \quad \theta^{(t+1)} := \theta^{(t)} - \frac{f(\theta^{(t)})}{f'(\theta^{(t)})}$$

The formula can be generalized to higher dimensions (which is needed here).

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

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

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

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

Remark. A stochastic/online version of gradient descent may be employed to increase speed.

How to classify new observations

After we fit the logistic model to the training set,

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

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

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

The general binary classification problem

When there are more than one predictor $x_1, \ldots, x_d$, just use

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

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

The classification rule also remains the same:

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

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

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

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

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

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).

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
```

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)

What is softmax regression?

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

$$\log \frac{P(Y=2 \mid \mathbf{x})}{P(Y=1 \mid \mathbf{x})} = \vec{\theta}_2 \cdot \mathbf{x}$$
$$\log \frac{P(Y=3 \mid \mathbf{x})}{P(Y=1 \mid \mathbf{x})} = \vec{\theta}_3 \cdot \mathbf{x}$$
$$\cdots$$
$$\log \frac{P(Y=c \mid \mathbf{x})}{P(Y=1 \mid \mathbf{x})} = \vec{\theta}_c \cdot \mathbf{x}$$

The prediction for a new observation will be the class with the largest relative probability.

Solving the system together with the constraint

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

yields that

$$P(Y = 1 \mid \mathbf{x}) = \frac{1}{1 + \sum_{j=2}^{c} e^{\vec{\theta}_j \cdot \mathbf{x}}}$$

and correspondingly,

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

Remarks:

- If we define $\vec{\theta}_1 = \mathbf{0}$, then the two sets of formulas may be unified

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

- We may relax the constant $\vec{\theta}_1$ to a parameter so that we may have a symmetric model, with (redundant) parameters $\Theta = \{\vec{\theta}_1, \ldots, \vec{\theta}_c\}$ each associated to a class.

- The distribution of Y, taking c values $1, \ldots, c$, is multinomial with the corresponding probabilities displayed above. Therefore, softmax regression is also called multinomial logistic regression.

Parameter estimation

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

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

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, 2DLDA) to project data into lower dimensions
- Add a **regularization** term to the objective function

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

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.

Summary

- Binary logistic regression
- Multiclass extensions
  - One versus one
  - One versus rest
  - Softmax/multinomial
- Regularized logistic regression

HW4 (due Friday noon, April 8)

This homework tests the logistic regression classifier on the MNIST digits. In Questions 1-4 below, apply PCA 50 to the digits first to reduce the dimensionality for logistic regression. In all questions below report your results using both graphs and texts.

1. Apply the binary logistic regression classifier to the following pairs of digits: (1) $0, 2$ (2) $1, 7$ and (3) $4, 9$.

2. Implement the one-versus-one extension of the binary logistic regression classifier and apply it to the MNIST handwritten digits.

3. Implement the one-versus-all extension of the binary logistic regression classifier and apply it to the MNIST handwritten digits.

4. Apply the multinomial logistic regression to the MNIST handwritten digits.

5. Apply the ℓ_1-regularized one-versus-all extension of binary logistic regression to the MNIST handwritten digits.

Midterm project 4: Logistic regression

Interested students please discuss with me your ideas.