San José State University

Math 251: Statistical and Machine Learning Classification

Logistic Regression

Dr. Guangliang Chen

Outline

- Introduction
- Logistic regression (2 classes)
  - Optimization perspective
  - Statistical perspective
- Softmax regression (3 classes or more)
- Regularized logistic regression
- Ordinal logistic regression
- Assignment 3 (cont'd)

Classification is a special instance of regression

Classification is a regression problem with a discrete response (i.e., a categorical variable taking a finite set of values):

$$\underbrace{y}_{\text{label}} \quad = \quad f\ (\underbrace{x_1, \ldots, x_d}_{\text{features}}).$$

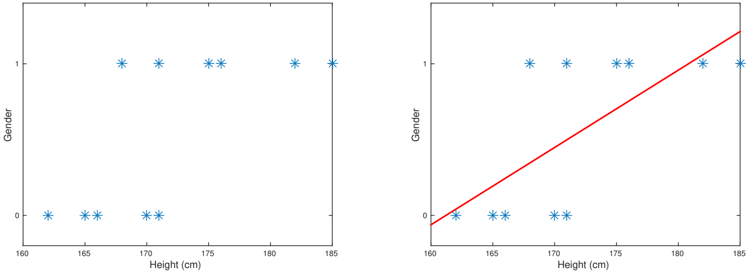
Thus, it also can be approached from a regression point of view.

To explain ideas, we start with a binary classification problem with only one feature:

$$\underbrace{y}_{\text{binary response}} \quad = \quad f(\underbrace{x}_{\text{1 predictor}}).$$

Motivating example

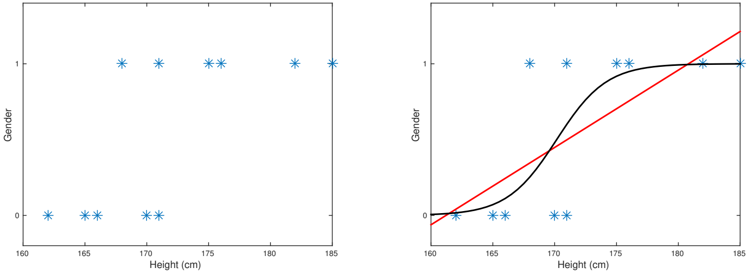
Consider a specific example where x represents a person's height while y denotes the person's sex ($0 =$ Female, $1 =$ Male).



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).

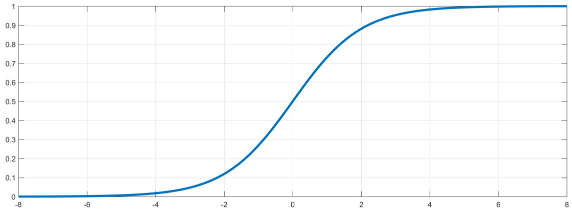


A better option is to use an S-shaped curve to fit the data.

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

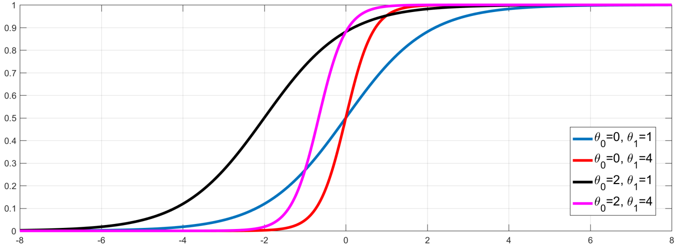
- $g(z)$ is defined for all real numbers z
- $g(z)$ is monotonically increasing over its domain
- $0 < g(z) < 1$ for all $z \in \mathbb{R}$
- $g(0) = 0.5$
- $\lim_{z\to-\infty} g(z) = 0$ and $\lim_{z\to+\infty} g(z) = 1$
- $g'(z) = g(z)(1 - g(z))$ for any z. $\longleftarrow$ This is a very important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

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.



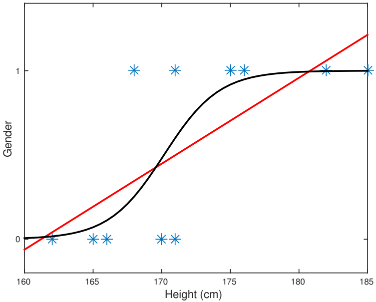
The logistic regression problem

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 $(x_i, y_i), 1 \le i \le n$, 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.

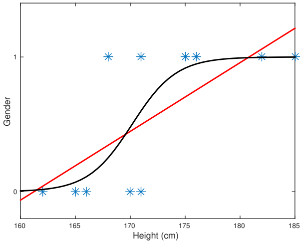
The optimization approach

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

The empirical loss of the parametric model on training data is defined as

$$\ell_n(\boldsymbol{\theta}; \{(x_i, y_i)\}) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \hat{y}_i)$$

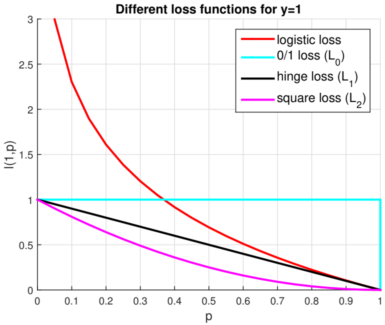
where $\hat{y}_i = g(\theta_0 + \theta_1 x_i)$.



The optimal values of $\boldsymbol{\theta}$ is found by minimizing ℓ_n.

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: $\ell(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$. Note that it imposes a much heavier penalty than the other losses when $\hat{y}$ is very wrong.



With a fixed loss function ℓ, the gradient of the empirical loss ℓ_n (as a function of θ_0, θ_1) is

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

The classical approach to minimizing ℓ_n is to find its critical points

$$\frac{1}{n} \sum_{i=1}^{n} \frac{\partial \ell}{\partial \theta_0}(y_i, \hat{y}_i) = 0, \quad \frac{1}{n} \sum_{i=1}^{n} \frac{\partial \ell}{\partial \theta_1}(y_i, \hat{y}_i) = 0$$

However, this often leads to very complex equations.

For example, with the square loss $\ell(y, \hat{y}) = (y - \hat{y})^2$, we have

$$\frac{\partial \ell}{\partial \theta_0} = 2(\hat{y} - y)\frac{\partial \hat{y}}{\partial \theta_0} = 2(\hat{y} - y) \cdot \hat{y}(1 - \hat{y})$$
$$\frac{\partial \ell}{\partial \theta_1} = 2(\hat{y} - y)\frac{\partial \hat{y}}{\partial \theta_1} = 2(\hat{y} - y) \cdot \hat{y}(1 - \hat{y}) \cdot x$$

The gradient of the empirical loss function ℓ_n is thus

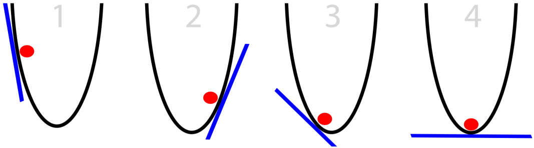
$$\frac{\partial \ell_n}{\partial \boldsymbol{\theta}} = \left(\frac{1}{n} \sum_{i=1}^{n} 2\hat{y}_i(1 - \hat{y}_i)(\hat{y}_i - y_i),\ \frac{1}{n} \sum_{i=1}^{n} 2x_i\hat{y}_i(1 - \hat{y}_i)(\hat{y}_i - y_i) \right)$$

Gradient descent

A modern, more efficient approach to finding the minimum of a function $f(x)$ is through **gradient descent**: Start from some initial location $x^{(0)}$ and update x iteratively as follows:

$$x^{(t+1)} = x^{(t)} - \eta \cdot f'(x^{(t)}), \quad t = 0, 1, 2, \ldots$$

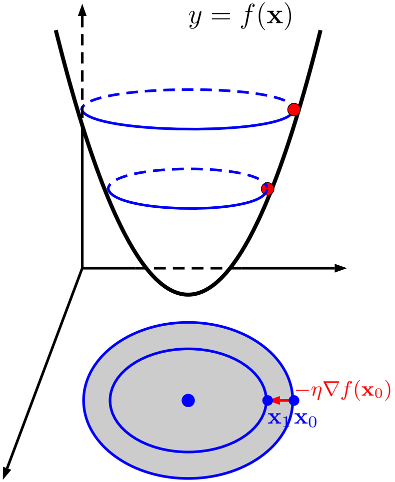
where $\eta > 0$ is a parameter, called *learning rate*.



This approach also applies to multivariate functions $f(\mathbf{x})$:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \cdot \nabla f(\mathbf{x}^{(t)}),$$

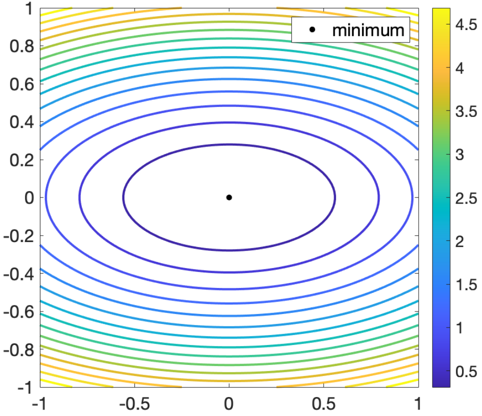
where the initial point $\mathbf{x}^{(0)}$ is specified by the user.



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

See next slide for an example.

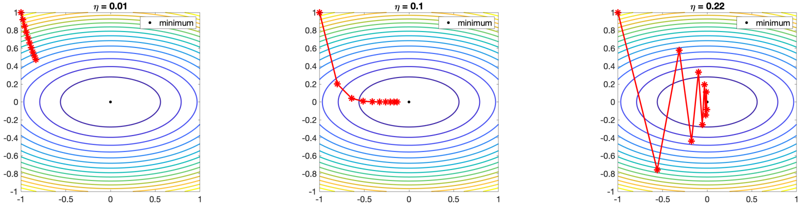
Example: Use gradient descent to find the minimum of $f(\mathbf{x}) = x_1^2 + 4x_2^2$

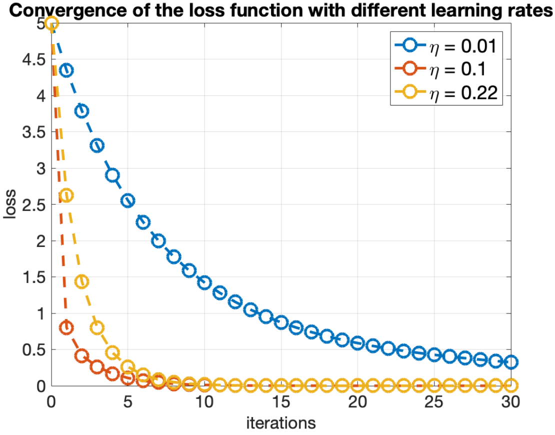


The gradient update rule is

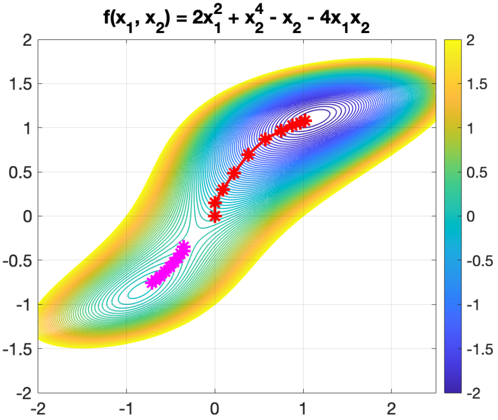
$$x_1^{(t+1)} \longleftarrow x_1^{(t)} - \eta \cdot 2x_1^{(t)}, \quad x_2^{(t+1)} \longleftarrow x_2^{(t)} - \eta \cdot 8x_2^{(t)}$$

Below shows the first 10 iterations of gradient descent with the same initialization $\mathbf{x}_0 = (-1, 1)$ but different learning rates:




Convergence of the loss function with different learning rates
η = 0.01
η = 0.1
η = 0.22
loss
iterations
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
5
10
15
20
25
30

Remark. Gradient descent only converges to a local minimum!



In the setting of logistic regression, the function to be optimized is $\ell_n(\boldsymbol{\theta})$. The gradient descent update rule is the following

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \cdot \left.\frac{\partial \ell_n}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}^{(t)}}, \quad t = 0, 1, 2, \ldots$$

where $\boldsymbol{\theta}^{(0)}$ is specified by the user.

The individual components of $\boldsymbol{\theta}$ are updated as follows:

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

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

With the square loss, the update rule is

$$\theta_0^{(t+1)} := \theta_0^{(t)} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} 2\hat{y}_i(1-\hat{y}_i)(\hat{y}_i - y_i)\Big|_{\theta_0^{(t)},\theta_1^{(t)}}$$

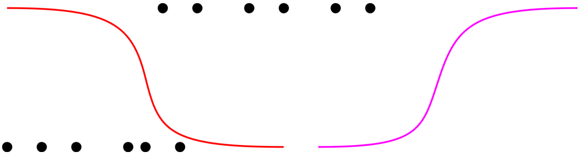
$$\theta_1^{(t+1)} := \theta_1^{(t)} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} 2x_i\hat{y}_i(1-\hat{y}_i)(\hat{y}_i - y_i)\Big|_{\theta_0^{(t)},\theta_1^{(t)}}$$

Despite the simplicity of the square loss, the update rule seems quite complicated.

Furthermore, the square loss based gradient descent does not work well in practice due to a so-called learning slowdown issue, which we explain next.

The learning slowdown issue due to square loss

Suppose the initial values $\theta_0^{(0)}, \theta_1^{(0)}$ are very wrong, such that the model is either curve:



In both cases, we have $\hat{y}_i(1-\hat{y}_i) \approx 0$ for almost all training points. Thus, $\frac{\partial \ell_n}{\partial \boldsymbol{\theta}} \approx \mathbf{0}$. This implies that both parameters would change little and gradient descent may need to spend a long time escaping either of the two wrong positions.

Fixing learning slowdown using the logistic loss

It turns out that with the use of the logistic loss (instead of square loss)

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

can avoid the learning slowdown issue.

To see this, compute first

$$\frac{\partial \ell}{\partial \theta_0} = -\frac{y}{\hat{y}} \cdot \hat{y}(1 - \hat{y}) + \frac{1 - y}{1 - \hat{y}} \cdot \hat{y}(1 - \hat{y}) = \hat{y} - y$$
$$\frac{\partial \ell}{\partial \theta_1} = -\frac{y}{\hat{y}} \cdot \hat{y}(1 - \hat{y})x + \frac{1 - y}{1 - \hat{y}} \cdot \hat{y}(1 - \hat{y})x = (\hat{y} - y)x$$

Thus, the gradient of the empirical loss is

$$\frac{\partial \ell_n}{\partial \boldsymbol{\theta}} = \left(\frac{1}{n}\sum_{i=1}^n (\hat{y}_i - y_i),\ \frac{1}{n}\sum_{i=1}^n x_i(\hat{y}_i - y_i) \right)$$

and correspondingly, the gradient descent update rule is

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

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

The term $\hat{y}_i(1-\hat{y}_i)$ is no longer in the partial derivations! Gradient descent will learn fast when many $\hat{y}_i$ are opposite to y_i.

Remark. The logistic loss has the following advantages:

- Forces $\hat{y}$ to be really close to y (by imposing a huge penalty otherwise)
- Leads to simple partial derivatives.
- Avoids learning slowdown (i.e., gradient descent will converge fast).
- Has a statistical interpretation, with connection to the MLE approach and Bayes classification (to be covered next)

Stochastic gradient descent (SGD)

Gradient descent requires access to the full training set, and uses all the training data simultaneously to update model parameters in each iteration.

This may be slow for large data sets, or impractical in the setting of online learning (where data comes sequentially).

A variant of gradient descent, called **stochastic gradient descent**, uses

- single training points, or
- small subsets of examples (called mini-batches),

to sequentially update model parameters in each iteration.

- **Single-point** update rule. In some random order ($i = 1, 2, \ldots$), pass the training points, one by one, to gradient descent in step t (called an **epoch**):

$$\theta_0^{(t)} \longleftarrow \theta_0^{(t)} - \eta \cdot \frac{\partial \ell}{\partial \theta_0}(y_i, \hat{y}_i)\Big|_{\theta_0^{(t)}, \theta_1^{(t)}}$$

$$\theta_1^{(t)} \longleftarrow \theta_1^{(t)} - \eta \cdot \frac{\partial \ell}{\partial \theta_1}(y_i, \hat{y}_i)\Big|_{\theta_0^{(t)}, \theta_1^{(t)}}$$

Once this epoch is completed, let $t = t + 1$ and move on to next epoch to take another pass through the training data.

- **Mini-batch** update rule. Divide the training data into (disjoint) mini-batches (of a fixed size such as 30), and pass them sequentially to gradient descent: For each mini-batch B, update

$$\theta_0^{(t)} \longleftarrow \theta_0^{(t)} - \eta \cdot \frac{1}{|B|} \sum_{i \in B} \frac{\partial \ell}{\partial \theta_0}(y_i, \hat{y}_i)\bigg|_{\theta_0^{(t)}, \theta_1^{(t)}}$$

$$\theta_1^{(t)} \longleftarrow \theta_1^{(t)} - \eta \cdot \frac{1}{|B|} \sum_{i \in B} \frac{\partial \ell}{\partial \theta_1}(y_i, \hat{y}_i)\bigg|_{\theta_0^{(t)}, \theta_1^{(t)}}$$

Once all the mini-batches have been used, an epoch is completed. We then let $t = t + 1$ and move on to next epoch to take another pass through the mini-batches.

In the case of the logistic loss, the single-point SGD update rule is

$$\begin{aligned}\theta_0^{(t)} &\longleftarrow \theta_0^{(t)} - \eta \cdot (\hat{y}_i - y_i)\big|_{\theta_0^{(t)},\theta_1^{(t)}} \\ \theta_1^{(t)} &\longleftarrow \theta_1^{(t)} - \eta \cdot x_i(\hat{y}_i - y_i)\big|_{\theta_0^{(t)},\theta_1^{(t)}}\end{aligned}$$

In contrast, the mini-batch SGD update rule is

$$\begin{aligned}\theta_0^{(t)} &\longleftarrow \theta_0^{(t)} - \eta \cdot \frac{1}{|B|}\sum_{i\in B}(\hat{y}_i - y_i)\Big|_{\theta_0^{(t)},\theta_1^{(t)}} \\ \theta_1^{(t)} &\longleftarrow \theta_1^{(t)} - \eta \cdot \frac{1}{|B|}\sum_{i\in B}x_i(\hat{y}_i - y_i)\Big|_{\theta_0^{(t)},\theta_1^{(t)}}\end{aligned}$$

Demonstration: linear regression via SGD

Consider the simple linear regression problem with a single predictor:

$$\min_{\hat{\boldsymbol{\beta}}=(\hat\beta_0,\hat\beta_1)} \quad S(\hat{\boldsymbol{\beta}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat\beta_0 - \hat\beta_1 x_i)^2$$

where $(x_i, y_i), 1 \le i \le n$ are training data.

The gradient of the loss function (mean squared error, or in short, MSE) is

$$\frac{\partial S}{\partial \hat\beta_0} = \frac{1}{n}\sum_{i=1}^{n} 2(\hat\beta_0 + \hat\beta_1 x_i - y_i),$$
$$\frac{\partial S}{\partial \hat\beta_1} = \frac{1}{n}\sum_{i=1}^{n} 2(\hat\beta_0 + \hat\beta_1 x_i - y_i)x_i$$

The full gradient descent update rule is

$$
\begin{aligned}
\hat{\beta}_0 &\longleftarrow \hat{\beta}_0 - \eta \cdot \frac{1}{n}\sum_{i=1}^{n} 2(\hat{\beta}_0 + \hat{\beta}_1 x_i - y_i),\\
\hat{\beta}_1 &\longleftarrow \hat{\beta}_1 - \eta \cdot \frac{1}{n}\sum_{i=1}^{n} 2x_i(\hat{\beta}_0 + \hat{\beta}_1 x_i - y_i)
\end{aligned}
$$

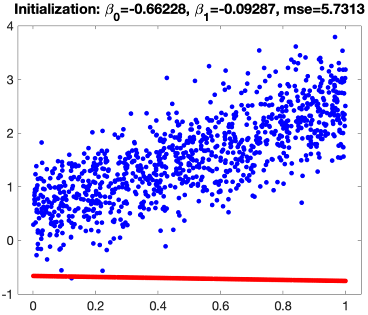
The SGD update rule (using mini-batches of size m) is

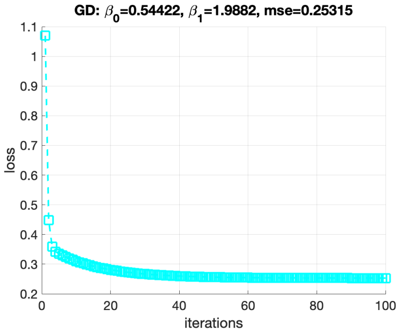
$$
\begin{aligned}
\hat{\beta}_0 &\longleftarrow \hat{\beta}_0 - \eta \cdot \frac{1}{m}\sum_{i\in B} 2(\hat{\beta}_0 + \hat{\beta}_1 x_i - y_i),\\
\hat{\beta}_1 &\longleftarrow \hat{\beta}_1 - \eta \cdot \frac{1}{m}\sum_{i\in B} 2x_i(\hat{\beta}_0 + \hat{\beta}_1 x_i - y_i)
\end{aligned}
$$

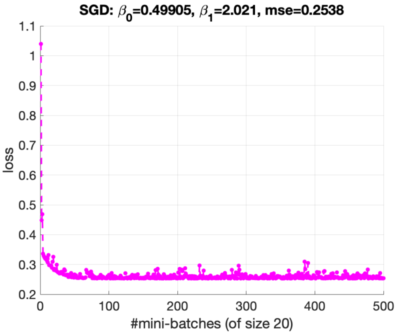
We draw a random sample of size 1000 from the line $y = \frac{1}{2} + 2x$ with additive Gaussian noise of mean 0 and standard deviation 0.5.

We perform both gradient descent and SGD (m=20) initialized with the red line. The learning rate is set to 0.5 for both methods.

See next slide for the convergence of the loss function with each method.





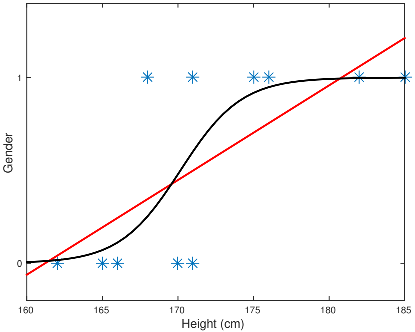


Remark.

- SGD is faster than full gradient descent, but maybe less stable
- SGD achieves a good balance between speed and stability

The statistical way

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



More specifically, consider the joint distribution of predictor X and (binary) response Y. Assume that

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

This implies that

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

with associated pmf

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

Given independently sampled training examples $(x_i, y_i), 1 \le i \le n$ from the joint distribution, the likelihood of the sample is

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

and the log likelihood is

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

The maximizer of $\log L(\boldsymbol{\theta})$, i.e., 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

Mathematically, the MLE formulation

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

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

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

So gradient descent can be used again to numerically solve the problem.

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_0:

$$\text{Assign label } y_0 = 1 \quad \text{if and only if} \quad p(x_0;\boldsymbol{\theta}) > \frac{1}{2}.$$

Remark. Logistic regression is also a Bayes classifier because it is based on the maximum posterior probability:

$$\underbrace{P(Y=1 \mid X=x,\boldsymbol{\theta})}_{p(x;\boldsymbol{\theta})} \quad > \quad \underbrace{P(Y=0 \mid X=x,\boldsymbol{\theta})}_{1-p(x;\boldsymbol{\theta})}$$

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
```

The general binary classification problem

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

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

where $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_d)$ and $\mathbf{x} = (x_0 = 1, x_1, \ldots, x_d)$.

We can use the same numerical methods 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 binary Logistic Regression (LR) classifier.

What kind of classifier is LR?

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

$$p(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{\theta} \cdot \mathbf{x}}} = \frac{1}{2}$$

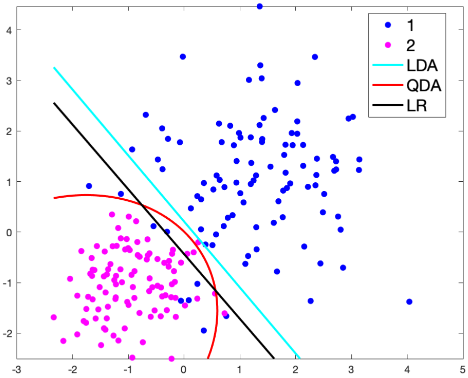
or equivalently,

$$\boldsymbol{\theta} \cdot \mathbf{x} = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d = 0$$

which is a hyperplane.

This shows that LR is a linear classifier.

Demonstration



LR is a generalized linear model (GLM)

The LR model

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

can be rewritten as

$$\text{link function (logit)} \longrightarrow \quad \log\frac{p}{1-p} = \boldsymbol{\theta}\cdot\mathbf{x}$$

where the response Y is a Bernoulli random variable with mean

$$\mathrm{E}(Y \mid \vec{X} = \mathbf{x};\boldsymbol{\theta}) = p(\mathbf{x};\boldsymbol{\theta})$$

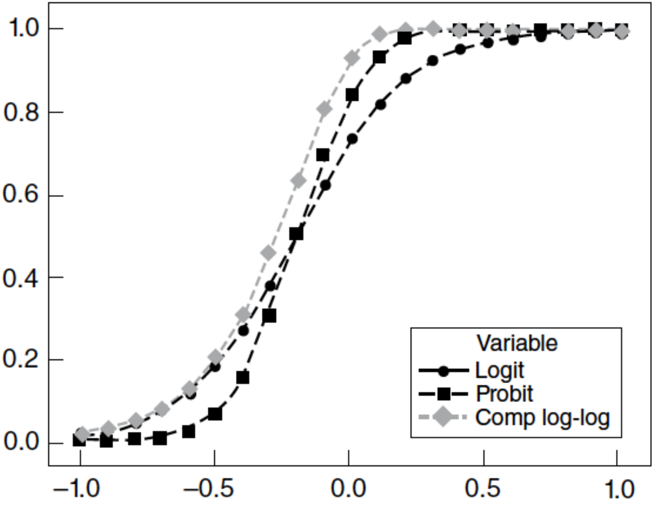
Other models for binary response data

Instead of using the logit link function which leads to the sigmoid function

$$\log \frac{p}{1-p} = \boldsymbol{\theta} \cdot \mathbf{x} \quad \longrightarrow \quad p(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{\theta} \cdot \mathbf{x}}}$$

one could use

- **Cauchit (inverse Cauchy)**: $\arctan(\pi(p - 0.5))$
- **Probit**: $\Phi^{-1}(p)$, where Φ is the cdf of standard normal.
- **Complimentary log-log**: $\log(-\log(1-p))$
- **Negative log-log**: $-\log(-\log(p))$


1.0
0.8
0.6
0.4
0.2
0.0
–1.0
–0.5
0.0
0.5
1.0
Variable
Logit
Probit
Comp log-log

Multiclass extensions

There are two ways to extend logistic regression for multiclass classification:

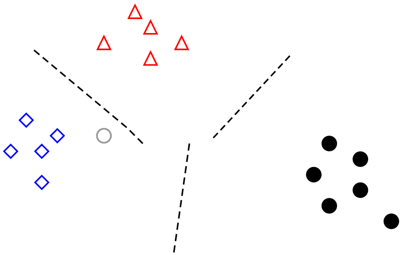
- **Union of binary models**
  - *One versus one*: construct a LR model for every pair of classes
  - *One versus rest*: construct a LR model for each class against the rest of the training set

  In either case, the prediction of the label of a test point is voted by all the models.

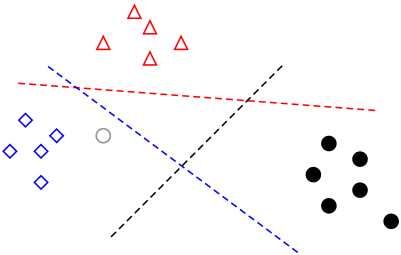
- **Softmax regression** (fixed versus rest)

Union of binary models

One-versus-one extension



One-versus-rest extension



How to determine the label of a test point $\mathbf{x}_0$:

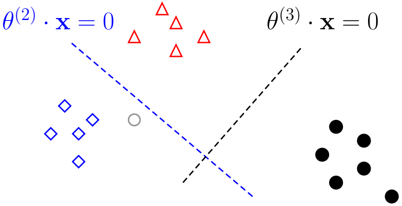
- For the one-versus-one extension, the final prediction for a test point is the majority vote by all the pairwise models;
- For the one-versus-rest extension,
  - Each reference class has new label 1 (the rest have label 0)
  - For each binary model with class j as the reference class, compute $p(\mathbf{x}_0, \boldsymbol{\theta}^{(j)})$
  - The final prediction is

$$\hat{y}_0 = \arg\max_j \; p(\mathbf{x}, \boldsymbol{\theta}^{(j)})$$

Softmax regression

...fixes one class (say class 1) and fits $c-1$ binary logistic regression models between each of the remaining classes $2 \le j \le c$ and class 1:

$$\log \frac{P(Y=j \mid \vec{X}=\mathbf{x})}{P(Y=1 \mid \vec{X}=\mathbf{x})} = \boldsymbol{\theta}^{(j)} \cdot \mathbf{x}$$



To better understand the method, write

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

We must have

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

It follows that

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

and

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

Letting $\boldsymbol{\theta}^{(1)} = \mathbf{0}$, we may unify the two sets of formulas

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

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

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

We may thus relax the fixed quantity $\boldsymbol{\theta}^{(1)} = \mathbf{0}$ to be a parameter in order to have a symmetric model:

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

with (redundant) parameters $\Theta = \{\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(c)}\}$.

Prediction for a new point $\mathbf{x}_0$ is based on the largest posterior probability:

$$\begin{aligned}
\hat{y}_0 &= \text{argmax}_{1 \le j \le c} P(Y = j \mid \vec{X} = \mathbf{x}_0) \\
&= \text{argmax}_j \; e^{\boldsymbol{\theta}^{(j)} \cdot \mathbf{x}_0} \\
&= \text{argmax}_j \; \boldsymbol{\theta}^{(j)} \cdot \mathbf{x}_0
\end{aligned}$$

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

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

It is a smooth function trying to approximate the indicator function

$$1_{\alpha_j = \max(\alpha_1, \ldots, \alpha_c)} = \begin{cases} 1, & \text{if } \alpha_j = \max(\alpha_1, \ldots, \alpha_c) \\ 0, & \text{otherwise} \end{cases}.$$

The conditional distribution of Y when given $\vec{X} = \mathbf{x}$ and Θ, is multinomial with probabilities $P(Y = j \mid \vec{X} = \mathbf{x}; \Theta), 1 \le j \le c$.

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

Parameter estimation

Given training data $\{(\mathbf{x}_i, y_i)\}_{1\le i\le n}$, softmax regression estimates the parameters by maximizing the likelihood of the training set:

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

Like before, 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 extraction/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 fix it:

- Use a **dimensionality reduction** method (such as PCA, LDA) to project data into low dimensions

- Add a **regularization** term to the objective function of the binary logistic regression classifier

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

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.

Regularized logistic regression in Matlab

```
% for two classes only

[B, stats] = lassoglm(Xtr, ytr, 'binomial','Link', 'logit', 'Lambda', C);

B0 = stats.Intercept;
coef = [B0; B];

yhat = glmval(coef, Xtst, 'logit');
ytsthat = (yhat>=0.5);
```

Regularized LR in Python

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.

- 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.

Logistic regression on the MNIST

See poster at

`https://www.sjsu.edu/faculty/guangliang.chen/Math285S16/poster-Logistic.pdf`

Ordinal logistic regression

Previously the response has been assumed to be (or treated as being) nominal.

In this part, we consider the case where there is a natural order among the response categories (i.e., ordinal responses): $\ell_1 < \ell_2 < \cdots < \ell_c$

The ordering might be

- inherent in the category choices, such as an individual being not satisfied, satisfied, or very satisfied with an online customer service.

- or introduced by categorization of a latent (continuous) variable, such as in the case of an individual being in the low risk, medium risk, or high risk group for developing a certain disease, based on a quantitative medical measure such as blood pressure.

For simplicity, we denote the labels by $1 < 2 < \cdots < c$ but note that this only indicates the ordering, the numbers are not equally spaced as they appear to be.

Ordinal models describe the relationship between cumulative probabilities of the categories $P(Y \leq j \mid \vec{X} = \mathbf{x})$ and predictor variables $\mathbf{x}$.

They are usually based on the assumption that **the effects of predictor variables are the same for all categories on the logarithmic scale**, but have different intercepts among different categories:

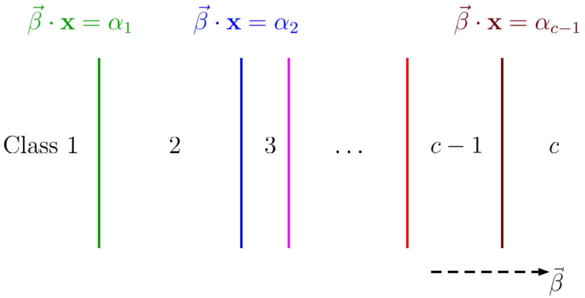
$$\log \frac{P(Y \leq j \mid \vec{X} = \mathbf{x})}{P(Y > j \mid \vec{X} = \mathbf{x})} = \alpha_j - \boldsymbol{\beta} \cdot \mathbf{x}, \quad j = 1, \ldots, c-1$$

where $\mathbf{x} = (x_1, \ldots, x_d)'$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_d)'$.

This model is called **parallel regression** or the **proportional odds** model.

Further understanding of the model:

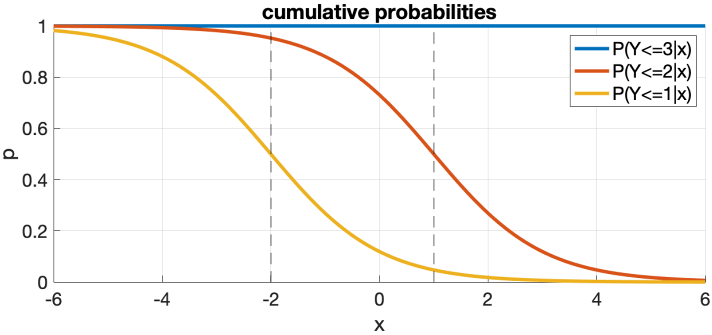
- β selects/combines features in the same way for all categories.
- Due to the setup, $\alpha_1 < \alpha_2 < \ldots < \alpha_{c-1}$ and they define the distances between the different categories along the direction of β



We can rewrite the model as

$$P(Y \leq j \mid \vec{X} = \mathbf{x}) = \frac{1}{1 + e^{-\alpha_j + \boldsymbol{\beta} \cdot \mathbf{x}}}, \quad j = 1, \ldots, c-1$$

Example: $c = 3, \alpha_1 = -2, \alpha_2 = 1, \beta = 1$

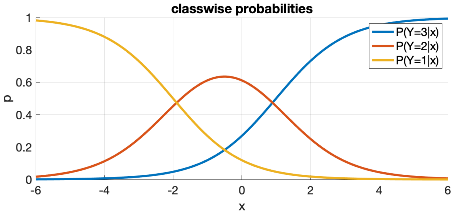


This implies that

$$P(Y=1 \mid \vec{X}=\mathbf{x}) = P(Y \le 1 \mid \vec{X}=\mathbf{x})$$

$$\begin{aligned} P(Y=j \mid \vec{X}=\mathbf{x}) &= P(Y \le j \mid \vec{X}=\mathbf{x}) - P(Y \le j-1 \mid \vec{X}=\mathbf{x}) \\ &= \frac{1}{1+e^{-\alpha_j+\boldsymbol{\beta}\cdot\mathbf{x}}} - \frac{1}{1+e^{-\alpha_{j-1}+\boldsymbol{\beta}\cdot\mathbf{x}}}, \quad 2 \le j \le c-1 \end{aligned}$$

$$P(Y=c \mid \vec{X}=\mathbf{x}) = 1 - P(Y \le c-1 \mid \vec{X}=\mathbf{x}) = 1 - \frac{1}{1+e^{-\alpha_{c-1}+\boldsymbol{\beta}\cdot\mathbf{x}}}$$



Remark. Parameter estimation is done via the MLE approach as before:

$$
\begin{aligned}
L(\boldsymbol{\alpha}, \boldsymbol{\beta} \mid \mathbf{x}_i, y_i) &= \prod_{i=1}^{n} P(Y = y_i \mid \mathbf{x}_i, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\
&= \prod_{j=1}^{c} \prod_{i:\, y_i = j} P(Y = j \mid \mathbf{x}_i, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\
&= \prod_{j=1}^{c} \prod_{i:\, y_i = j} \left(\frac{1}{1 + e^{-\alpha_j + \boldsymbol{\beta} \cdot \mathbf{x}_i}} - \frac{1}{1 + e^{-\alpha_{j-1} + \boldsymbol{\beta} \cdot \mathbf{x}_i}} \right)
\end{aligned}
$$

Remark. The previous ordinal regression model uses the logit link function for each binary model

$$\log \frac{P(Y \leq j \mid \vec{X} = \mathbf{x})}{P(Y > j \mid \vec{X} = \mathbf{x})} = \alpha_j - \boldsymbol{\beta} \cdot \mathbf{x}, \quad j = 1, \ldots, c-1$$

We could use other link functions instead, such as the probit:

$$\Phi^{-1}(P(Y \leq j \mid \vec{X} = \mathbf{x})) = \alpha_j - \boldsymbol{\beta} \cdot \mathbf{x}, \quad j = 1, \ldots, c-1$$

This will lead to a so-called **ordered probit** model.

Matlab implementation

```
B = mnrfit(Xtr,ytr,'model','ordinal','interactions', 'off', 'link', 'logit')
% default model is nominal
```

```
cumP = mnrval(B,Xtst,'type','cumulative','model','ordinal','interactions','off');
```

Assignment 3 (cont'd)

3 Apply the three multiclass extensions of the binary logistic regression classifier (one-vs-one, one-vs-rest, and multinomial) to the Fashion-MNIST data set (after PCA 95%). How do they compare with the multiclass LDA classifier in terms of test accuracy and running time?

4 Apply the ℓ_1-regularized multinomial logistic regression to the FashionMNIST data set (no pca is needed). How does it compare with those methods in Question 3?