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):

\[
y_{\text{label}} = f(x_1, \ldots, x_d).
\]

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:

\[
y_{\text{binary response}} = f(x_{\text{1 predictor}}).
\]
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.
Logistic Regression

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.
Logistic Regression

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 $z$. This is a very important property, implying that $g'(z) \approx 0$ for $z$ in either tail.
Logistic Regression

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.
Logistic Regression

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 $\theta_0, \theta_1$ 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.
Logistic Regression

**How to define the optimality**

There are two different ways:

- **Optimization** approach.

- **Statistical** approach.
Logistic Regression

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:

$$\min_{\theta=(\theta_0, \theta_1)} \ell_n(\theta) \overset{\text{def}}{=} \sum_{i=1}^{n} \ell(y_i, \hat{y}_i),$$

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

In the above,

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

Clearly, the previous formulation is independent of the choices of the template function \( g \) and of the loss function \( \ell \).

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**: \( \ell(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \)
Computing tool: multivariable calculus

The gradient of the total loss $\ell_n$ (as a function of $\theta_0, \theta_1$) is

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

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, with the square loss $\ell(y, \hat{y}) = (y - \hat{y})^2$, the gradient is

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

Proof:
Finding the critical points in this case is highly nontrivial:

\[
\sum_{i=1}^{n} \hat{y}_i (1 - \hat{y}_i) (\hat{y}_i - y_i) = 0, \quad \sum_{i=1}^{n} x_i \hat{y}_i (1 - \hat{y}_i) (\hat{y}_i - y_i) = 0
\]

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 $x^{(0)}$ and always move by a small amount along the direction of largest decrease (i.e., negative gradient):

Update rule:

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

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

This approach can also be generalized to higher dimensions:

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

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

In our setting, the gradient descent rule is the following

\[
\begin{align*}
\theta_0^{(t+1)} & := \theta_0^{(t)} - \eta \sum_{i=1}^{n} \left. \frac{\partial \ell}{\partial \theta_0} (y_i, \hat{y}_i) \right|_{\theta_0^{(t)}, \theta_1^{(t)}} \\
\theta_1^{(t+1)} & := \theta_1^{(t)} - \eta \sum_{i=1}^{n} \left. \frac{\partial \ell}{\partial \theta_1} (y_i, \hat{y}_i) \right|_{\theta_0^{(t)}, \theta_1^{(t)}}
\end{align*}
\]
With the square loss, the update rule is

\[
\theta_0^{(t+1)} := \theta_0^{(t)} - 2\eta \sum_{i=1}^{n} \hat{y}_i (1 - \hat{y}_i)(\hat{y}_i - y_i)
\]

\[
\theta_1^{(t+1)} := \theta_1^{(t)} - 2\eta \sum_{i=1}^{n} x_i \hat{y}_i (1 - \hat{y}_i)(\hat{y}_i - y_i)
\]

Despite the simplicity of the square loss, it does not work well in practice due to the learning slowdown issue: If the initialization of \(\theta_0, \theta_1\) is very wrong such that \(\hat{y}_i\) is in the opposite direction of \(y_i\), it will take a long time for \(\hat{y}_i\) to escape that part and move toward \(y_i\).
Logistic Regression

It turns out that with the logistic loss $\ell(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log (1 - \hat{y})$, the gradient is

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

and correspondingly, the gradient descent update rule is

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

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

This has fixed the learning slowdown!
Remark. The logistic loss has the following advantages:

- Forces $\hat{y}$ to be really close to $y$ (by imposing a huge penalty otherwise).
- 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).
The statistical way

Key idea: Interpret \( p(x, \theta) = g(\theta_0 + \theta_1 x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}} \) as probability!
More specifically, consider the joint distribution of predictor $X$ and (binary) response $Y$. Assume that

\[
P(Y = 1 \mid X = x, \theta) = p(x; \theta),
\]
\[
P(Y = 0 \mid X = x, \theta) = 1 - p(x; \theta)
\]

This implies that

\[
(Y \mid X = x, \theta) \sim \text{Bernoulli}(p = p(x; \theta))
\]

with associated pmf

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

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

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

and the log likelihood is

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

The maximizer of \(\log L(\theta)\), i.e., the Maximum Likelihood Estimator (MLE) of \(\theta\), 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

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

is equivalent to the following minimization problem

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

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 $\theta$

This falls under the previous setting of optimization via multivariable calculus in two ways (this and next two slides are essentially a repetition but for completeness, they are still included).

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

$$
\left( \frac{\partial \log L(\theta)}{\partial \theta_0}, \frac{\partial \log L(\theta)}{\partial \theta_1} \right) = \left( \sum_{i=1}^{n} (y_i - p(x_i; \theta)), \sum_{i=1}^{n} (y_i - p(x_i; \theta)) x_i \right)
$$
The same two methods can be used to find the MLE:

- **Critical-point** method:

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

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

Due to the complex form, *Newton’s iteration* has to be used.
Logistic Regression

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

\[
\begin{align*}
\theta_0^{(t+1)} & := \theta_0^{(t)} + \eta \cdot \sum_{i=1}^{n} (y_i - p(x_i; \theta^{(t)})) \\
\theta_1^{(t+1)} & := \theta_1^{(t)} + \eta \cdot \sum_{i=1}^{n} (y_i - p(x_i; \theta^{(t)})) x_i
\end{align*}
\]

in which \(\eta > 0\) is the *learning rate*.
# How to classify new observations

After we fit the logistic model to the training set,

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

we may use the following decision rule for a new observation \( x_0 \):

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

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

\[
\frac{P(Y = 1 \mid X = x, \theta)}{p(x; \theta)} > \frac{P(Y = 0 \mid X = x, \theta)}{1 - p(x; \theta)}
\]
MATLAB functions for logistic regression

```matlab
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]
```
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, 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

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

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

We can use the same numerical methods to find the best $\theta$.

The classification rule also remains the same:

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

We call this classifier the binary Logistic Regression (LR) classifier.
What kind of classifier is LR?

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

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

or equivalently,

$$\theta \cdot x = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d = 0$$

which is a hyperplane.

This shows that LR is a linear classifier.
Demonstration
Logistic Regression

**LR is a generalized linear model (GLM)**

The LR model

\[
p(x; \theta) = \frac{1}{1 + e^{-\theta \cdot x}}
\]

can be rewritten as

\[
\text{link function (logit)} \quad \rightarrow \quad \log \frac{p}{1 - p} = \theta \cdot x
\]

where the response \( Y \) is a Bernoulli random variable with mean

\[
E(Y \mid \vec{X} = x; \theta) = p(x; \theta)
\]
Other models for binary response data

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

\[
\log \frac{p}{1 - p} = \theta \cdot x \quad \rightarrow \quad p(x; \theta) = \frac{1}{1 + e^{-\theta \cdot x}}
\]

one could use

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

Union of binary models

- One-versus-one extension
- One-versus-rest extension
Logistic Regression

How to determine the label of a test point $x_0$:

- 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 $p(x_0, \theta^{(j)})$
  
  - The final prediction is

$$\hat{y}_0 = \arg\max_j p(x, \theta^{(j)})$$
Softmax regression

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

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

To better understand the method, write

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

We must have

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

It follows that

\[ P(Y = 1 \mid \vec{X} = x) = \frac{1}{1 + \sum_{2 \leq j \leq c} e^{\theta(j) \cdot x}} \]

and

\[ P(Y = j \mid \vec{X} = x) = \frac{e^{\theta(j) \cdot x}}{1 + \sum_{2 \leq \ell \leq c} e^{\theta(\ell) \cdot x}}, \quad j = 2, \ldots, c \]
Letting $\theta^{(1)} = 0$, we may unify the two sets of formulas

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

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

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

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

\[
P(Y = j \mid \vec{X} = \vec{x}; \Theta) = \frac{e^{\theta^{(j)} \cdot \vec{x}}}{\sum_{1 \leq \ell \leq c} e^{\theta^{(\ell)} \cdot \vec{x}}}, \quad j = 1, \ldots, c
\]

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

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

\[
\hat{y}_0 = \arg\max_{1 \leq j \leq c} P(Y = j \mid \vec{X} = \vec{x}_0)
\]

\[
= \arg\max_{j} e^{\theta^{(j)} \cdot \vec{x}_0}
\]

\[
= \arg\max_{j} \theta^{(j)} \cdot \vec{x}_0
\]
**Remark.** The form of the posterior probabilities is the so-called softmax function:

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

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 $\tilde{X} = x$ and $\Theta$, is multinomial with probabilities $P(Y = j \mid \tilde{X} = x; \Theta), 1 \leq j \leq c$.

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

Given training data \( \{(x_i, y_i)\}_{1 \leq i \leq n} \), softmax regression estimates the parameters by maximizing the likelihood of the training set:

\[
L(\Theta) = \prod_{i=1}^{n} P(Y = y_i \mid \vec{X} = x_i; \Theta) = \prod_{i=1}^{n} \frac{e^{\theta(y_i) \cdot x_i}}{\sum_{1 \leq j \leq c} e^{\theta(j) \cdot x_i}}
\]

Like before, the MLE can be found by using either Newton’s method or gradient descent.
MATLAB functions for multinomial LR

\[
x = [162 \ 165 \ 166 \ 170 \ 171 \ 168 \ 171 \ 175 \ 176 \ 182 \ 185]';
\]
\[
y = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]';
\]
\[
B = mnrfit(x,\text{categorical}(y));
\]
\[
p = mnrval(B, x);
\]
Logistic Regression

Python function for multinomial LR

```python
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')
```
Logistic Regression

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
Logistic Regression

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

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

where \( p \) is normally set to 2 (\( \ell_2 \) regularization) or 1 (\( \ell_1 \) 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);
Logistic Regression

Regularized LR in Python

```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

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 $P(Y \leq j \mid \vec{X} = \vec{x})$ and predictor variables $\vec{x}$.

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:

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

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

This model is called parallel regression or the proportional odds model.
Further understanding of the model:

- \( \beta \) 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 \( \beta \).

\[
\begin{align*}
\vec{\beta} \cdot \mathbf{x} &= \alpha_1 \\
\vec{\beta} \cdot \mathbf{x} &= \alpha_2 \\
\vec{\beta} \cdot \mathbf{x} &= \alpha_{c-1}
\end{align*}
\]
Logistic Regression

We can rewrite the model as

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

**Example:** $c = 3$, $\alpha_1 = -2$, $\alpha_2 = 1$, $\beta = 1$
This implies that

\[ P(Y = 1 \mid \vec{X} = \vec{x}) = P(Y \leq 1 \mid \vec{X} = \vec{x}) \]

\[ P(Y = j \mid \vec{X} = \vec{x}) = P(Y \leq j \mid \vec{X} = \vec{x}) - P(Y \leq j - 1 \mid \vec{X} = \vec{x}) \]

\[ = \frac{1}{1 + e^{-\alpha_j + \beta \cdot x}} - \frac{1}{1 + e^{-\alpha_{j-1} + \beta \cdot x}}, \quad 2 \leq j \leq c - 1 \]

\[ P(Y = c \mid \vec{X} = \vec{x}) = 1 - P(Y \leq c - 1 \mid \vec{X} = \vec{x}) = 1 - \frac{1}{1 + e^{-\alpha_{c-1} + \beta \cdot x}} \]
Remark. Parameter estimation is done via the MLE approach as before:

\[
L(\alpha, \beta \mid x_i, y_i) = \prod_{i=1}^{n} P(Y = y_i \mid x_i, \alpha, \beta) = \prod_{j=1}^{c} \prod_{i: y_i = j} P(Y = j \mid x_i, \alpha, \beta) = \prod_{j=1}^{c} \prod_{i: y_i = j} \left( \frac{1}{1 + e^{-\alpha_j + \beta \cdot x_i}} - \frac{1}{1 + e^{-\alpha_{j-1} + \beta \cdot x_i}} \right)
\]
Remark. The previous ordinal regression model uses the logit link function for each binary model

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

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

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

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

Matlab implementation

\[ B = \text{mnrfit}(Xtr, ytr, 'model', 'ordinal', 'interactions', 'off', 'link', 'logit') \]

% default model is nominal

\[ \text{cumP} = \text{mnrval}(B, Xtst, 'type', 'cumulative', 'model', 'ordinal', 'interactions', 'off'); \]
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 $\ell_1$-regularized multinomial logistic regression to the FashionMNIST data set (no pca is needed). How does it compare with those methods in Question 3?