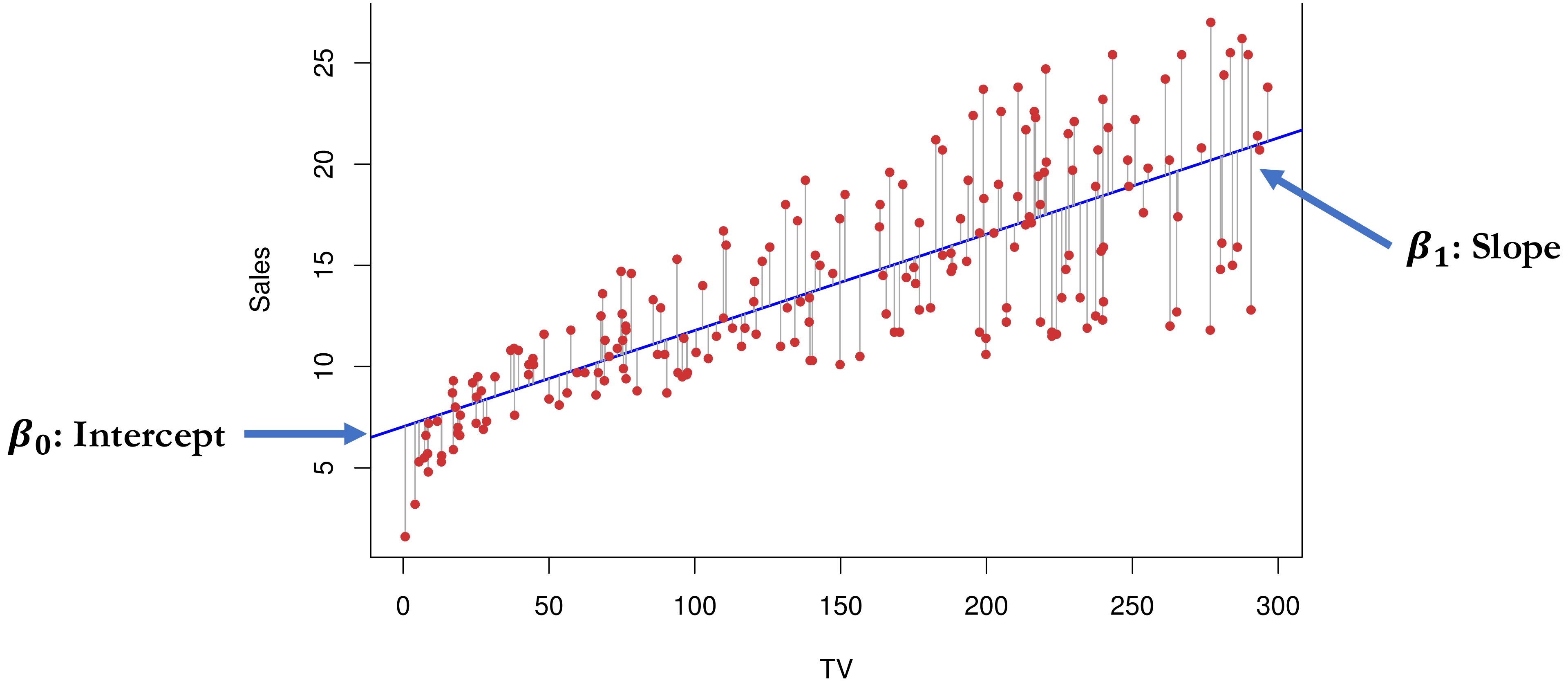
Introduction to Artificial Intelligence

Lecture 3: Supervised learning II

September 11, 2025

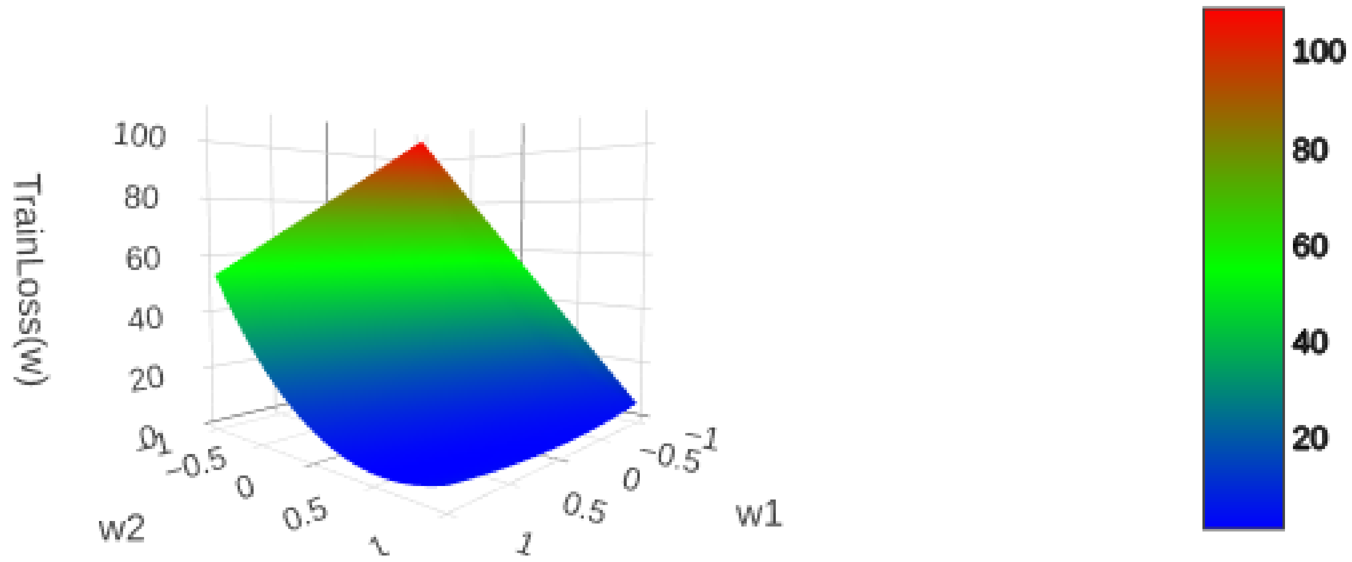
Recap: Linear regression framework

- Given examples as training data, design a learning algorithm to learn a predictor that maps unseen input to an output value



Loss function: How good is a predictor?

- Hypothesis class: which predictor to choose from?
  - Design a loss function. Question: which loss function?



Optimization algorithm

- Goal: minimize the training loss over β
- Definition (gradient): let $f: \mathbb{R}^d \to \mathbb{R}$ be a multi-dimensional function, which takes a vector of d variables X as input, and outputs a real value $y = f(X)$. Suppose f is differentiable at every coordinate, then, the gradient of f, denoted as ∇f, is defined as

$$\nabla f(X) = \begin{bmatrix} \frac{\partial f(X)}{\partial X_1}, \\ \frac{\partial f(X)}{\partial X_2}, \\ \cdots, \\ \frac{\partial f(X)}{\partial X_d} \end{bmatrix}$$

Computing the gradient

- Objective function:

$$\text{TrainingLoss}(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left(x_i^\top \beta - y_i\right)^2$$

- Gradient of the loss (use chain rule):

$$\nabla_\beta \text{TrainingLoss}(\beta) = \frac{1}{n}\sum_{i=1}^{n} 2\left(x_i^\top \beta - y_i\right)x_i$$

Boston housing dataset example

- Boston housing dataset:
  - CRIM: per capita crime rate by town
  - ZN: proportion of residential land zoned for lots over 25,000 sqft
  - INDUS: proportion of non-retail business acres per town
  - CHAS: Charles River dummy variable (1 if tract bounds river; 0 otherwise)
  - NOX: nitric oxides concentration (parts per 10 million)
  - RM: average number of rooms per dwelling
  - AVG: proportion of owner-occupied units built prior to 1940
  - DIS: weighted distances to five Boston employment centers
  - RAD: index of accessibility to radial highways
  - TAX: full-value property-tax rate per $10,000
  - PTRATIO: pupil-teacher ratio by town
  - B-1000: (Bk - 0.63)^2 where Bk is the proportion of blacks by town
  - LSTAT: % lower status of the population
  - <u>MEDV: Median value of owner-occupied homes in $1000's</u>

Gradient descent in python

- Loading the dataset into python

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
# Any results you write to the current directory are saved as output.
from pandas import read_csv
#Lets load the dataset and sample some
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MED
data = read_csv('./housing.csv', header=None, delimiter=r"\s+", names=column_names)

from sklearn import preprocessing
# Let's scale the columns before plotting them against MEDV
min_max_scaler = preprocessing.MinMaxScaler()
column_sels = ['LSTAT', 'INDUS', 'NOX', 'PTRATIO', 'RM', 'TAX', 'DIS', 'AGE']
x = data.loc[:,column_sels]
y = data['MEDV']
x = pd.DataFrame(data=min_max_scaler.fit_transform(x), columns=column_sels)

y =  np.log1p(y).values.reshape(-1, 1)
for col in x.columns:
    if np.abs(x[col].skew()) > 0.3:
        x[col] = np.log1p(x[col])
```

Gradient descent in python

```
# Training using gradient descent
m, n = x.shape

use_bias = True
if use_bias:
    X = np.hstack([np.ones((m, 1)), x])
    theta = np.zeros((n + 1, 1))
else:
    X = x
    theta = np.zeros((n, 1))


learning_rate = 0.1
iters = 5000

for _ in range(iters):
    y_pred = X @ theta
    grad = (X.T @ (y_pred - y)) / m
    theta -= learning_rate * grad

train_mse = np.mean((X @ theta - y) ** 2)
print("training set MSE:", train_mse)
```

```
training set MSE: 0.03838560047396505
```

Stochastic gradient descent in python

```
# Training using mini-batch gradient descent
m, n = x.shape

use_bias = True
if use_bias:
    X = np.hstack([np.ones((m, 1)), x])
else:
    X = x

rng = np.random.default_rng(42)
idx = rng.permutation(m)
batch_size = 32
batches = [idx[i:i+batch_size] for i in range(0, m, batch_size)]
theta = np.zeros((n + 1, 1))
iters = 5000
learning_rate = 0.1

for epoch in range(iters):
    for batch in batches:
        X_batch = X[batch]
        y_batch = y[batch]
        y_pred_batch = X_batch @ theta
        grad = (X_batch.T @ (y_pred_batch - y_batch)) / X_batch.shape[0]
        theta -= learning_rate * grad

y_pred = X @ theta
train_mse = np.mean((y_pred - y) ** 2)
print("training set MSE:", train_mse)
```

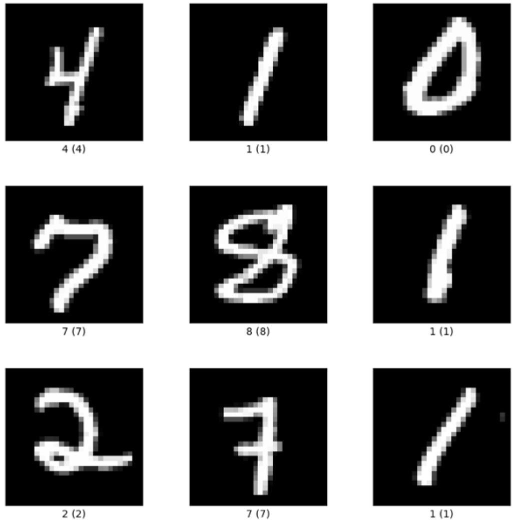
```
training set MSE: 0.03709784026839069
```

Lecture plan

- **Supervised learning**
  - **Linear classification**

Classification example

- Handwritten digit classification
- Colored handwritten digits
- Street view house numbers



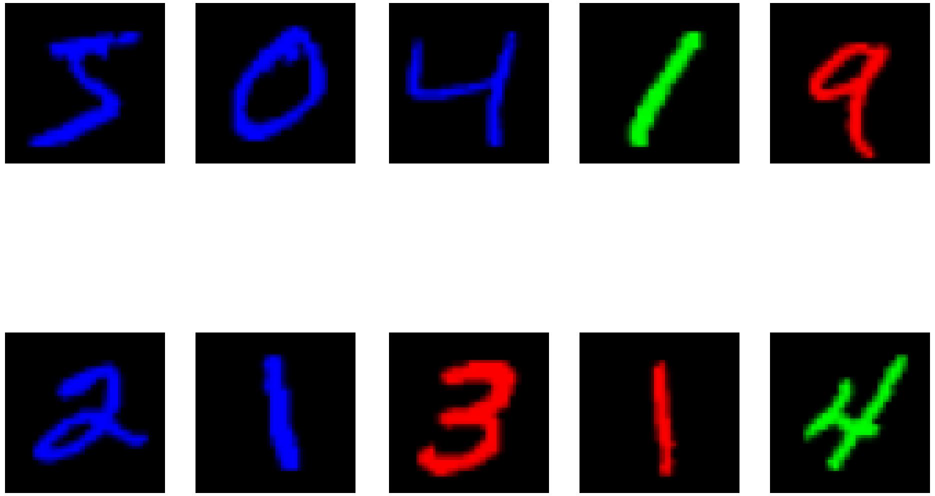
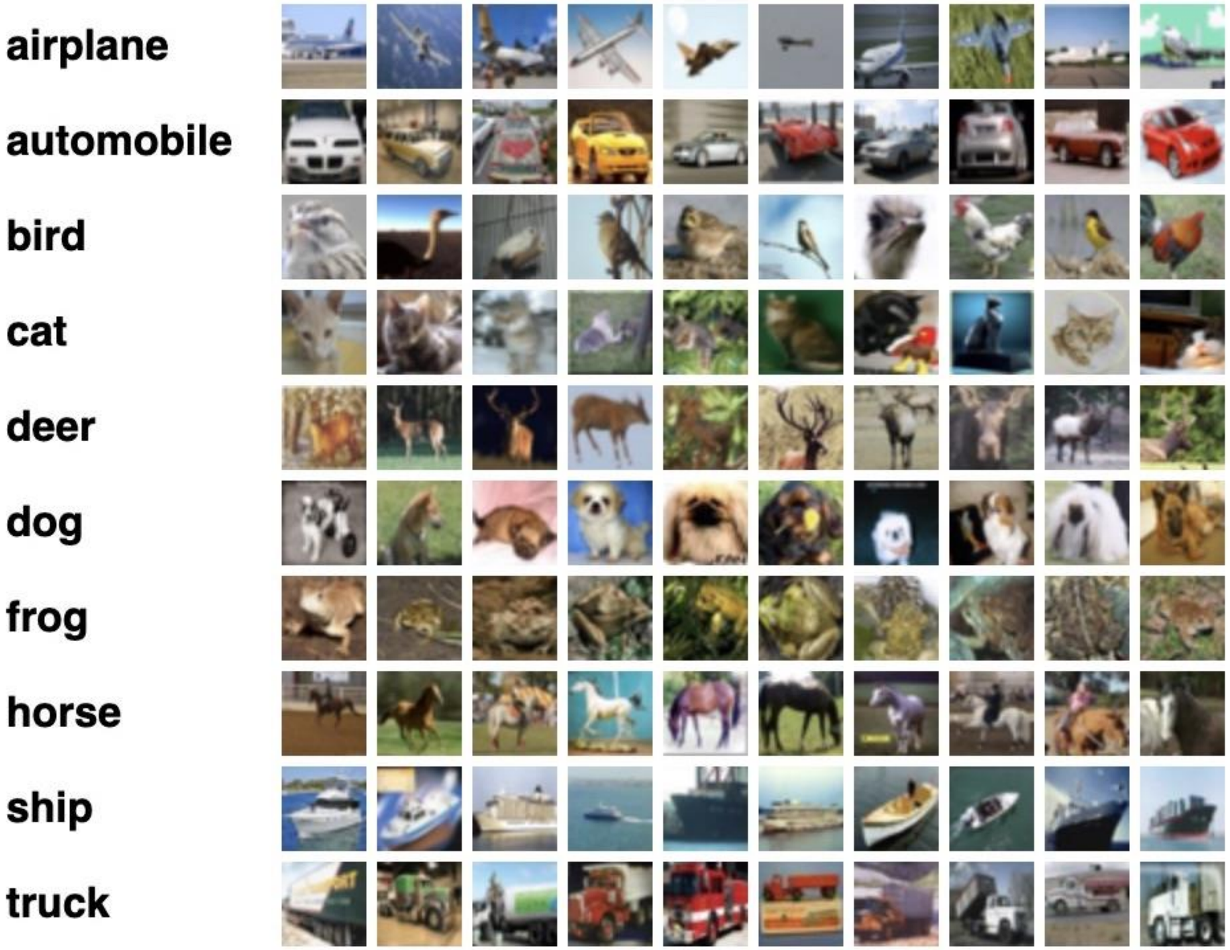


Image classification

- Image classification: assign a label to an entire image or photograph
- Object recognition



CIFAR-10: Canadian Institute For Advanced Research 60,000 images in 10 different classes, with 6,000 images of each class

Binary classification

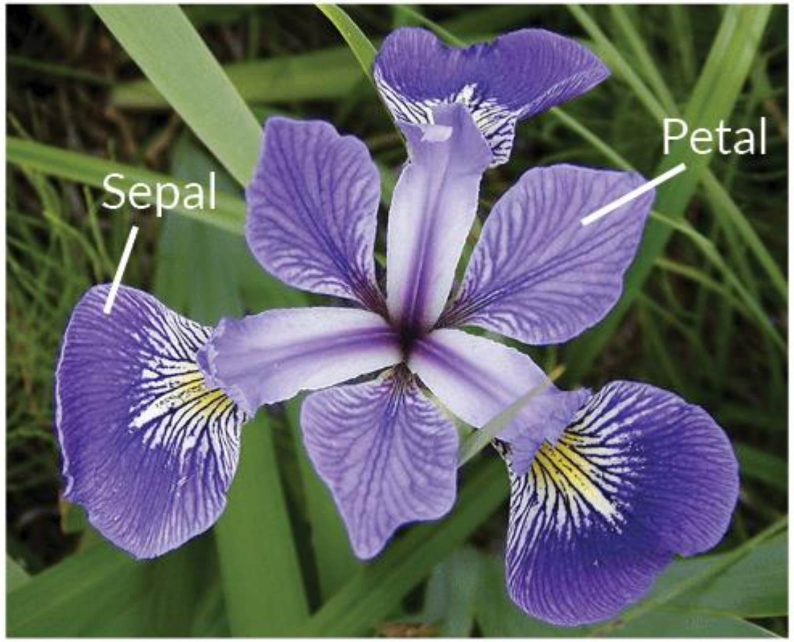
- Information on ten thousand customers
  - **default**: whether the customer defaulted on their debt
  - **student**: whether the customer is a student
  - **balance**: the average balance that the customer has remaining on their credit card after making their monthly payment
  - **income**: income of customer

- Predict which customers will default on their credit card debt

```
## # A tibble: 10,000 x 4
##    default student balance income
##    <fct>   <fct>     <dbl>  <dbl>
##  1 No      No         730. 44362.
##  2 No      Yes        817. 12106.
##  3 No      No        1074. 31767.
##  4 No      No         529. 35704.
##  5 No      No         786. 38463.
##  6 No      Yes        920.  7492.
##  7 No      No         826. 24905.
##  8 No      Yes        809. 17600.
##  9 No      No        1161. 37469.
## 10 No      No           0  29275.
## # … with 9,990 more rows
```

Iris dataset

- Pattern recognition: Predict class of iris plant. There are three classes



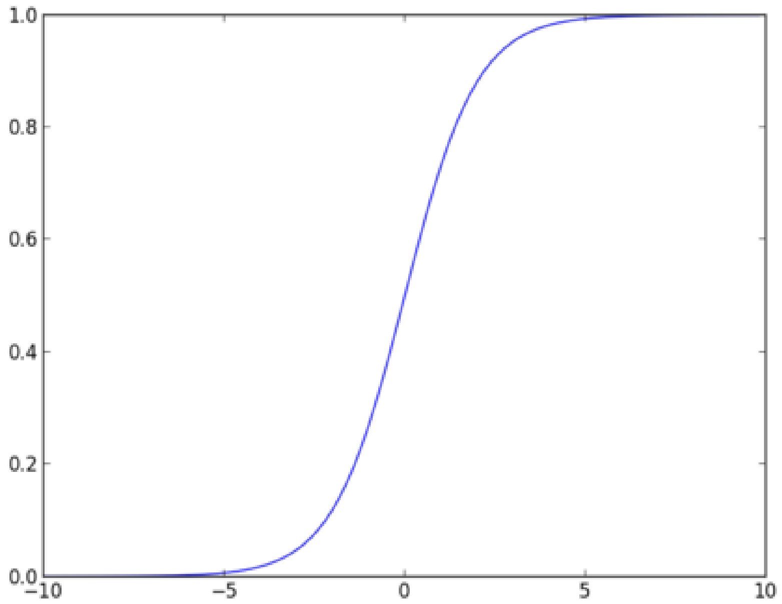
Iris Versicolor

Iris Setosa

Iris Virginica

Logistic function

- Zero-one loss: Loss value is zero if predicted label is correct, is one otherwise
- Logistic loss provides an approximation of the zero-one loss
  - Stanford logistic function: $\frac{1}{1+e^{-v}}$



Logistic loss

- Insert v from a linear model: $\frac{\exp(v)}{1+\exp(v)}$, still ranging between zero and one

- Logistic loss: negative log of logistic function

$$\ell(v) = -\log \frac{\exp(v)}{1 + \exp(v)} = \log(1 + \exp(-v))$$

- Question: Is $\ell(10)$ positive or negative? What about $\ell(-10)$?

Logistic regression

- Suppose the labels are either $+1$ or -1
  - For every sample x_i, y_i, suppose x_i includes p features in total, indexed by subscripts as $x_{i,1}, x_{i,2}, \ldots, x_{i,p}$
  - Coefficients of the logistic regression model: $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$. Let

$$v_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_p x_{i,p}$$

- The log-loss of x_i, y_i is

$$\log(1 + \exp(-y_i \cdot v_i))$$

- Averaged training loss over a dataset of size n

$$\frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i \cdot v_i))$$

Log probability

- The logit odds of a sample X, Y is

$$\log\left[\frac{Pr(Y=1|X)}{Pr(Y=0|X)}\right] = \beta^\top X = \beta_0 + \beta_1 \cdot \text{student} + \beta_2 \cdot \text{balance} + \beta_3 \cdot \text{income},$$

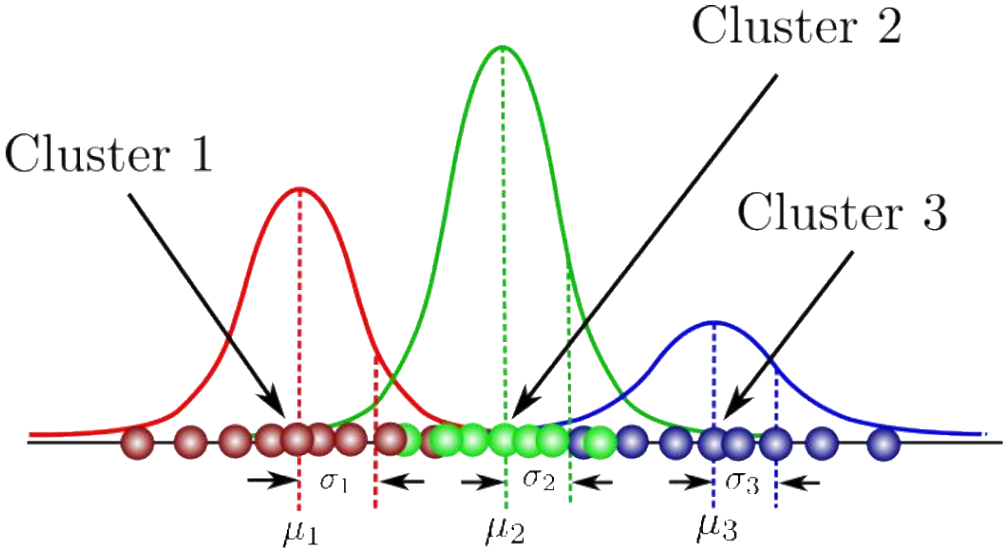
where $X = (1, student, balance, income)$, and

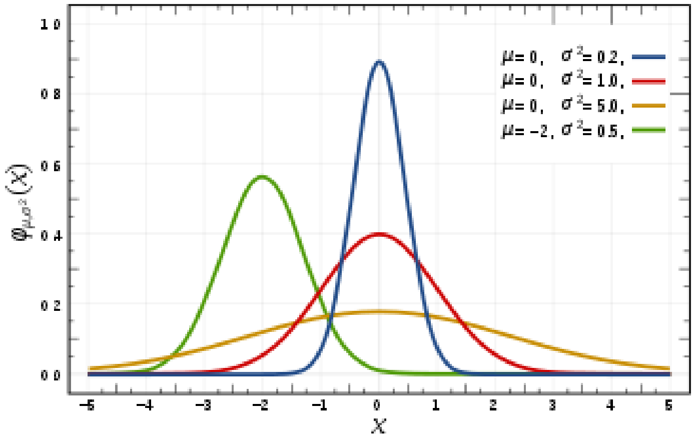
$$\Pr[Y=1|X] = \frac{1}{1+e^{-(\beta_0+\beta_1\cdot\text{student}+\beta_2\cdot\text{balance}+\beta_3\cdot\text{income})}}$$

- Logistic regression: Find $\beta_0, \beta_1, \beta_2, \beta_3$ by minimizing the averaged log-loss over the training set
- Prediction: if $v_i > 0$, $\hat{y}_i = +1$; if $v_i \leq 0$, $\hat{y}_i = -1$

Example

- **Mixture of Gaussians**: Crab measurements of forehead to body length ratio among 1000 crabs





Reference: http://blog.mrtz.org/2014/04/22/pearsons-polynomial.html

Linear classifier

Suppose we have K classes, we approximate the data distribution of each class with a Gaussian distribution

- π_k: prior probability that a randomly chosen observation comes from the k-th class
- $f_k(X) = \Pr(X|Y = k)$: density function of X coming from the k-th class
- $\Pr(Y = k|X = x)$: probability of x having label k

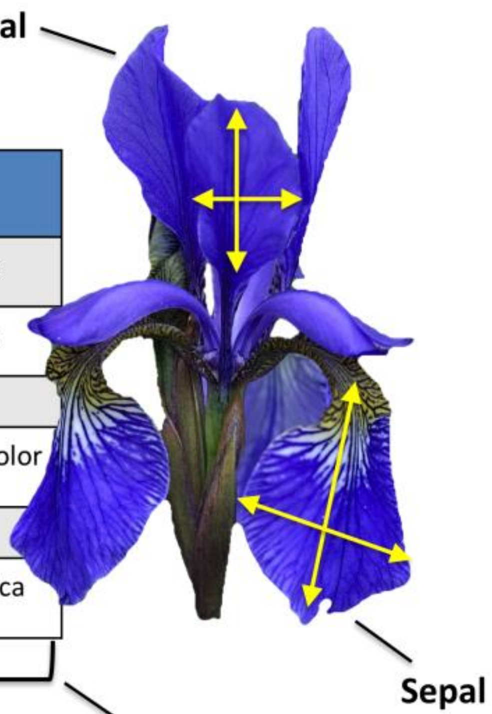
Sepal and petal of iris

Example: Iris dataset

- 50 samples from each of three classes of *Iris (versicolor, setosa, virginica)*
- Four features: sepal length, sepal width, petal length, petal width

Samples
(instances, observations)

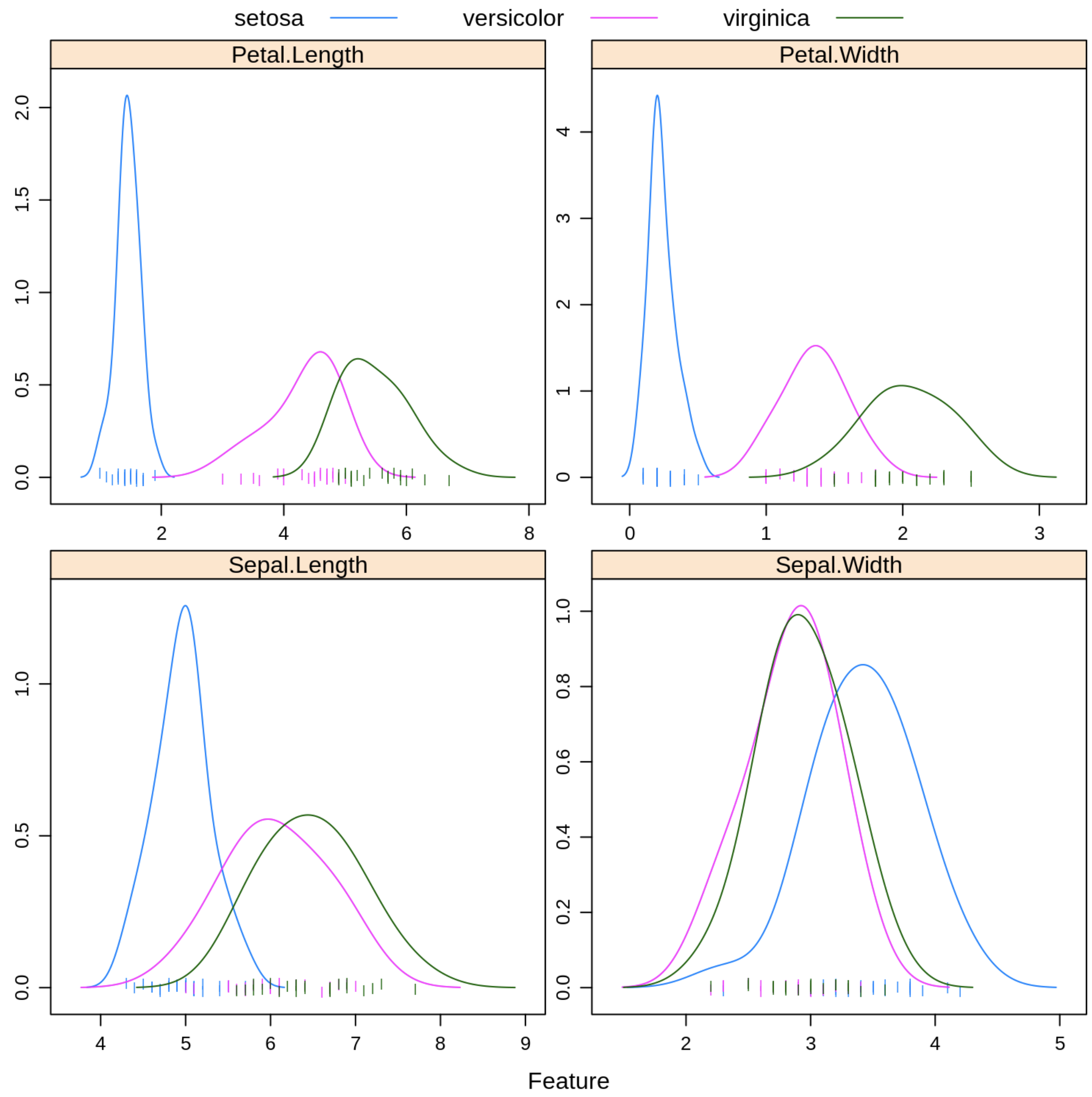
	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica



Class labels
(targets)

Features
(attributes, measurements, dimensions)

Distribution of features



Iris Versicolor **Iris Setosa** **Iris Virginica**

Generative model: Linear discriminant analysis

- Model $\Pr[X = x \mid Y = k]$

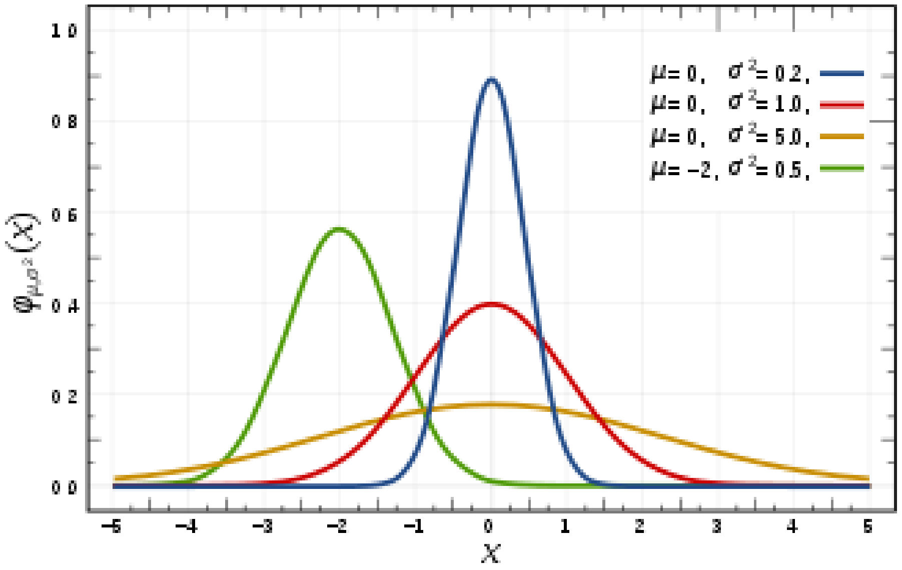
$$X = \begin{bmatrix} \text{sepal length} \\ \text{sepal width} \\ \text{petal length} \\ \text{petal width} \end{bmatrix}$$

$$Y \in \{\text{versicolor}, \text{setosa}, \text{virginica}\}$$

by a **multivariate normal distribution** $N(\mu_k, \Sigma)$ with mean μ_k, covariance matrix Σ

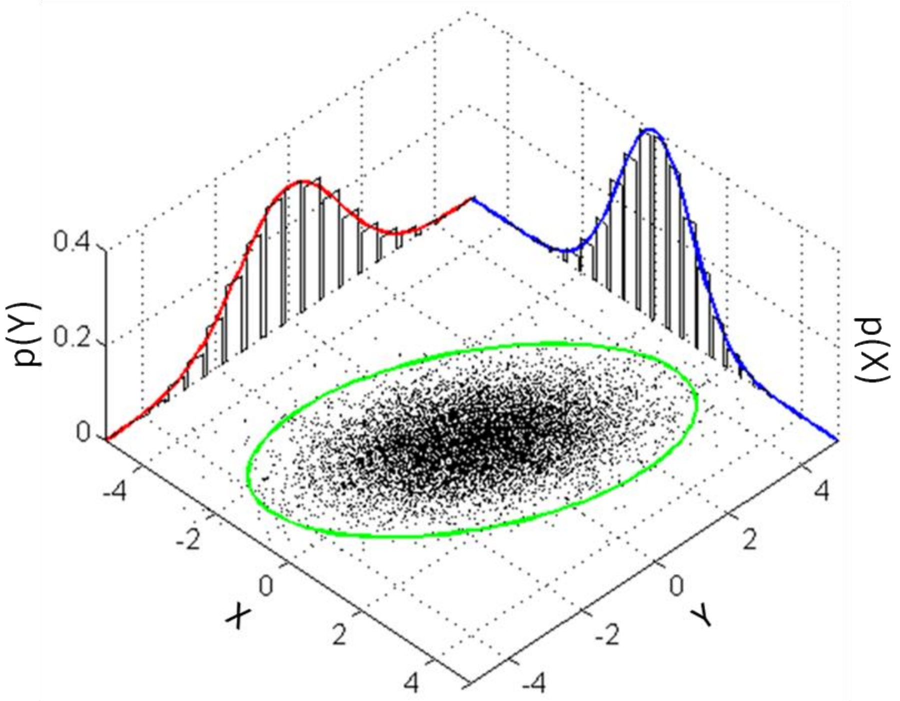
One-dimensional data

- For the k-th class, model density function as $N(\mu_k, \sigma^2)$
- Density function: $\Pr[X = x|Y = k] = f_k(x) = \frac{1}{\sqrt{2\pi\sigma_k}} \exp\left(-\frac{1}{2\sigma_k^2}(x - \mu_k)^2\right)$
- Within each class, the features have a center μ_k for every class k and **common variance $\boldsymbol{\sigma^2}$**



Multi-dimensional case

- $N(\mu, \Sigma)$ is a multi-dimensional Gaussian with mean μ, covariance Σ: μ is a p-dimensional vector, covariance is a $p \times p$ matrix: $\Sigma = E[xx^\top]$
- Illustration of a two-dimensional multivariate normal distribution
- Two dimensions: blue and red
- Projection to every dimension is still a Gaussian
- Centered at zero



Multi-dimensional data

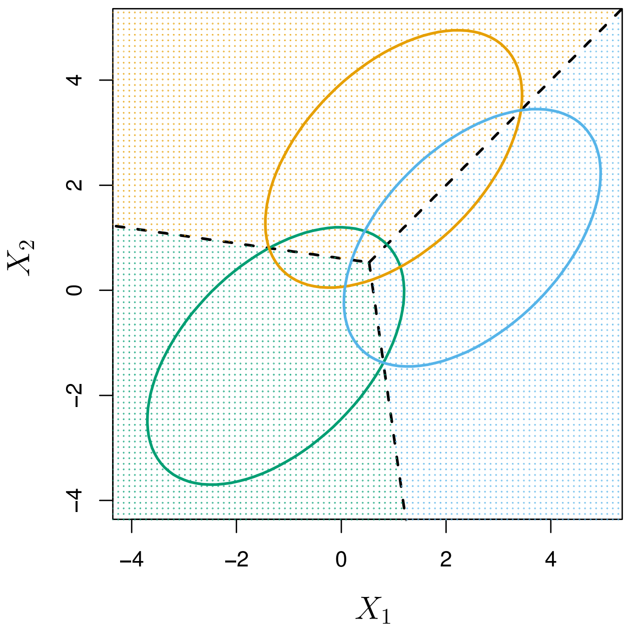
- For the k-th class, model density function as $N(\mu_k, \Sigma)$
- Density function

$$\Pr[X = x | Y = k] = f_k(x) = \frac{1}{(2\pi)^{\frac{p}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \mu_k)^\top \Sigma (x - \mu_k)\right)$$

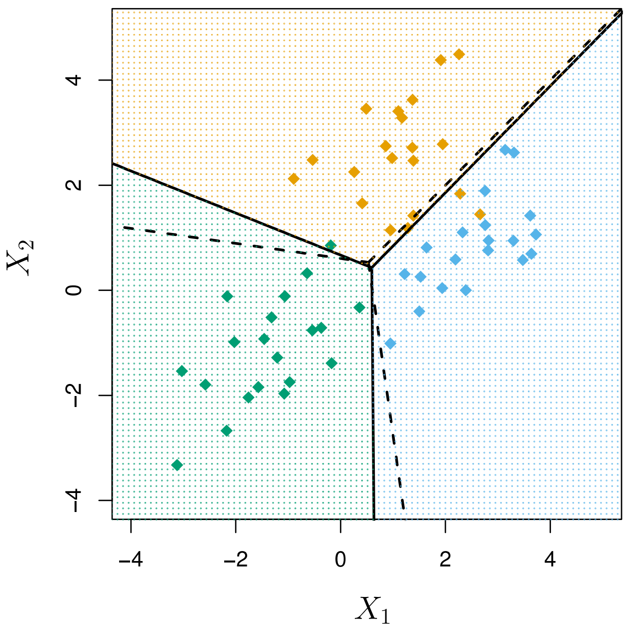
- Within each class, the features have a center μ_k for every class k and **common variance $\boldsymbol{\sigma^2}$**

Example

- Example with a two-dimensional synthetic dataset



Dash lines: Bayes decision boundaries



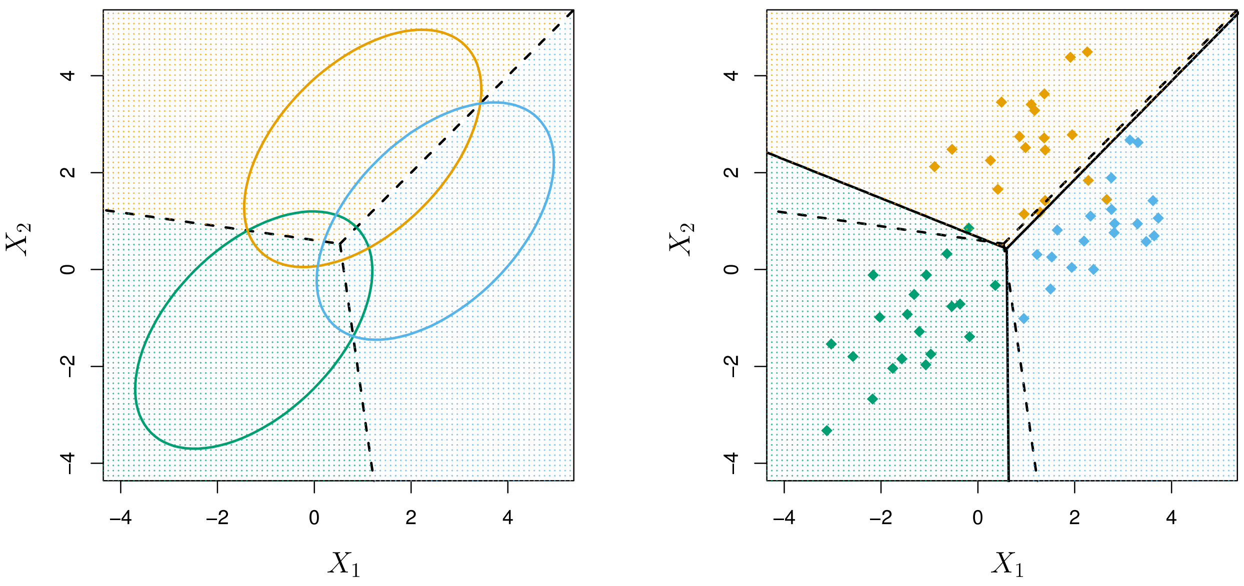
Solid lines: LDA decision boundaries (they are linear)

Estimating the center

How does this work?

1. Estimate the center of each class μ_k:

$$\hat{\mu}_k = \frac{1}{\#\{i: y_i = k\}} \sum_{i: y_i = k} x_i$$

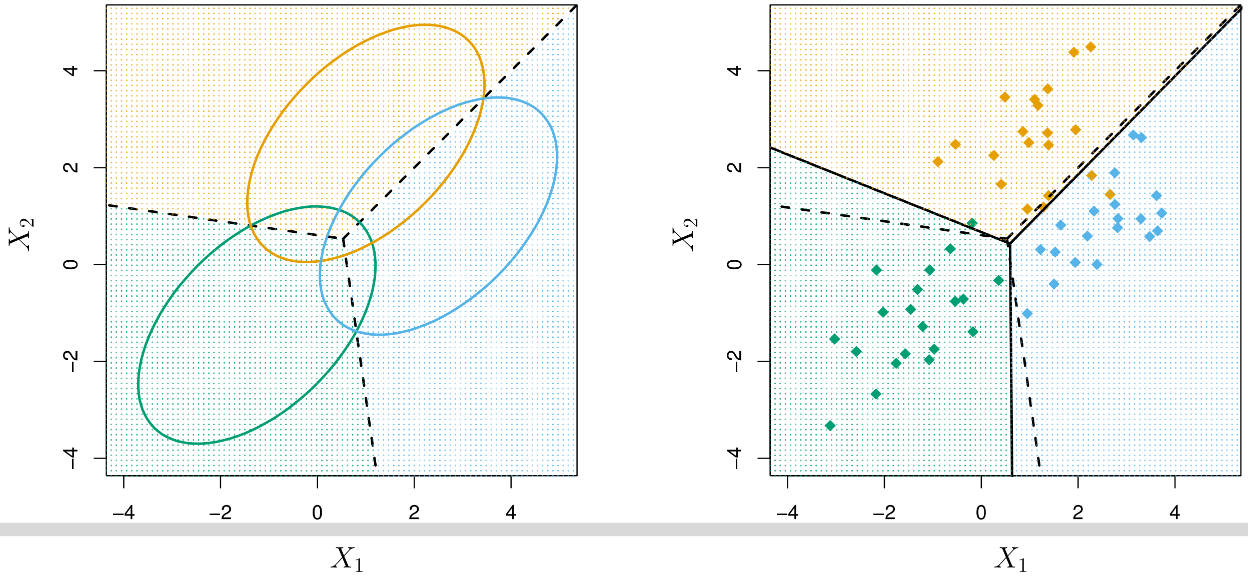


Estimating the covariance

How does this work?

2. Estimate the common covariance matrix Σ

- One-dimensional data: $\hat{\sigma}^2 = \frac{1}{n}\sum_{k=1}^{K}\sum_{i:y_i=k}(x_i-\hat{\mu}_k)^2$
- Multi-dimensional data: Compute the vectors of deviations $(x_1-\hat{\mu}_{y_1}), (x_2-\hat{\mu}_{y_2}), \cdots, (x_n-\hat{\mu}_{y_n})$ and their covariance

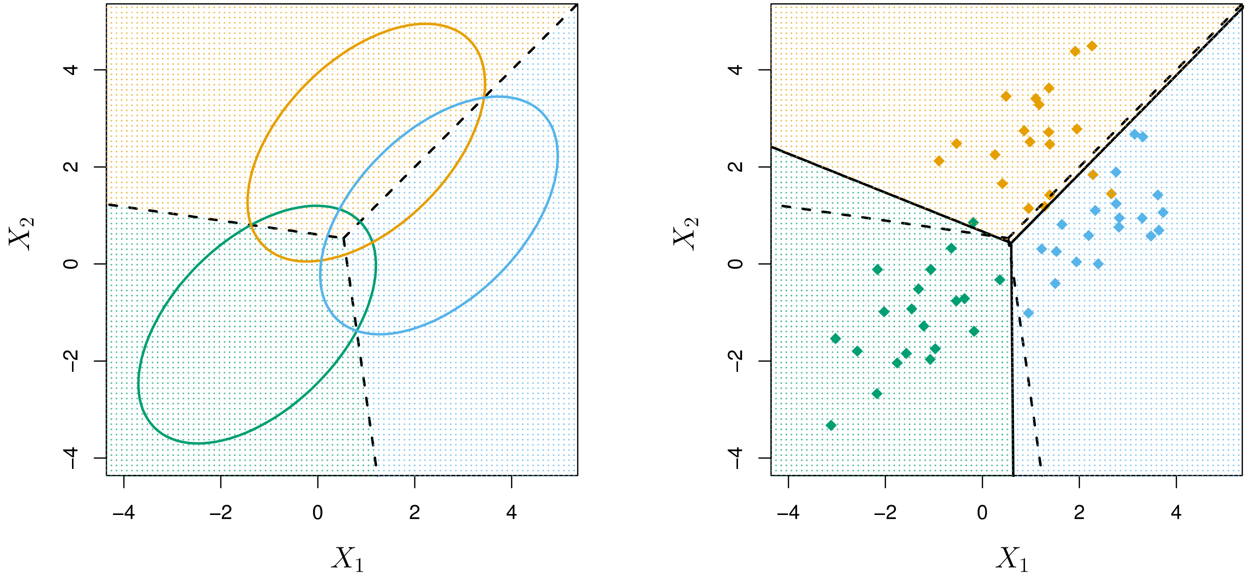


Estimating the prior

How does this work?

3. Estimated by the fraction of training samples of class k: $\Pr[Y = k] = \hat{\pi}_k$

$$\hat{\pi}_k = \frac{\#\{i : y_i = k\}}{n}\text{: Fraction of training samples of class } k$$



Prediction

- Recall: $\Pr(Y = k|X = x)$ is probability of x having label k
- LDA predicts the label with highest probability
- Bayes rule

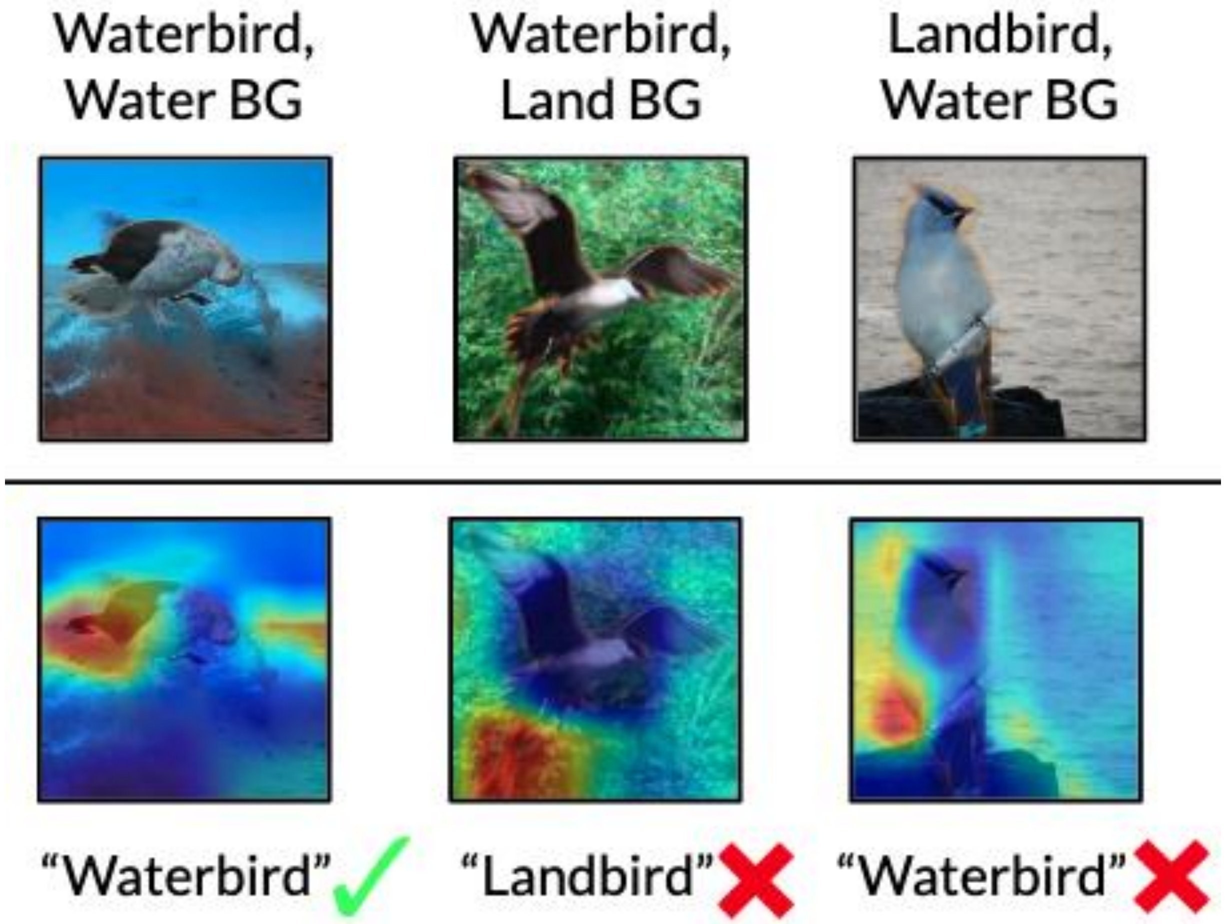
$$\Pr[Y = k|X = x] = \frac{Pr(Y = k, X = x)}{Pr(X = x)} = \frac{Pr(X = x|Y = k) \cdot Pr(Y = k)}{\sum_{i=1}^{K} Pr(X = x|Y = i) \cdot Pr(Y = i)}$$

Lecture plan

- Supervised learning
  - Linear regression
  - Linear classification
  - **Group robust regression**

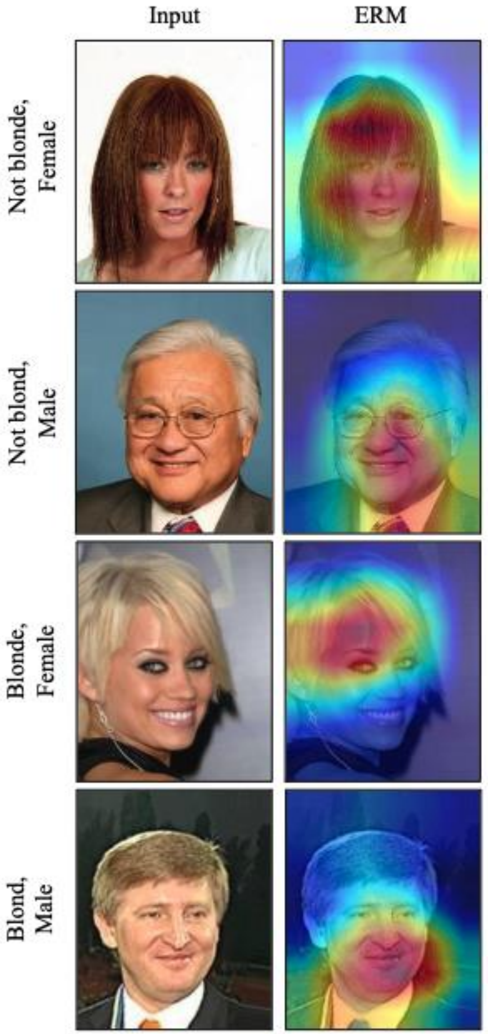
Spurious correlations

- Classifying water bird vs. land bird



Spurious correlations

- The Celebrities dataset: strong correlation between blonde hair color with female gender



Linear regression with groups

- Suppose we have G groups, $1,2,\ldots,G$, in total
  - In the previous example: {water bird, water background}, {water bird, land background}, {land bird, water background}, {land bird, land background}
- The predictor does not use group information g
- To encode this prior into the model, we could introduce a per-group loss

$$\hat{L}_g(\beta) = \frac{1}{n_g} \sum_{(x,y)\in D_g} \ell(x, y; \beta)$$

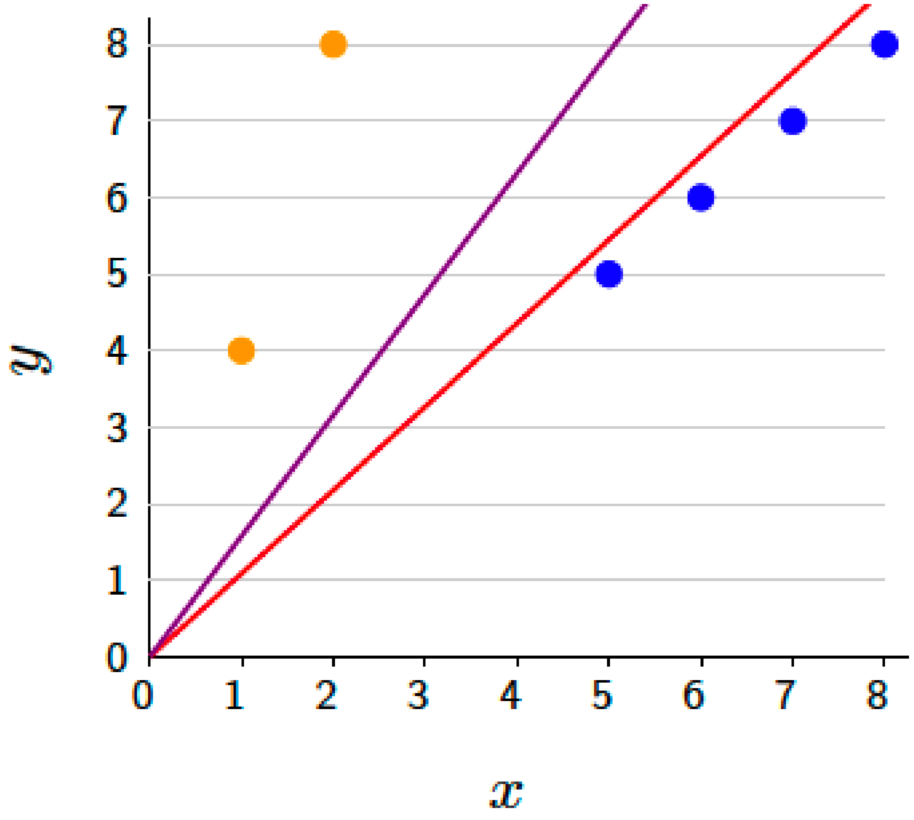
- Instead of minimizing the average loss, we could now minimize the maximum group loss instead

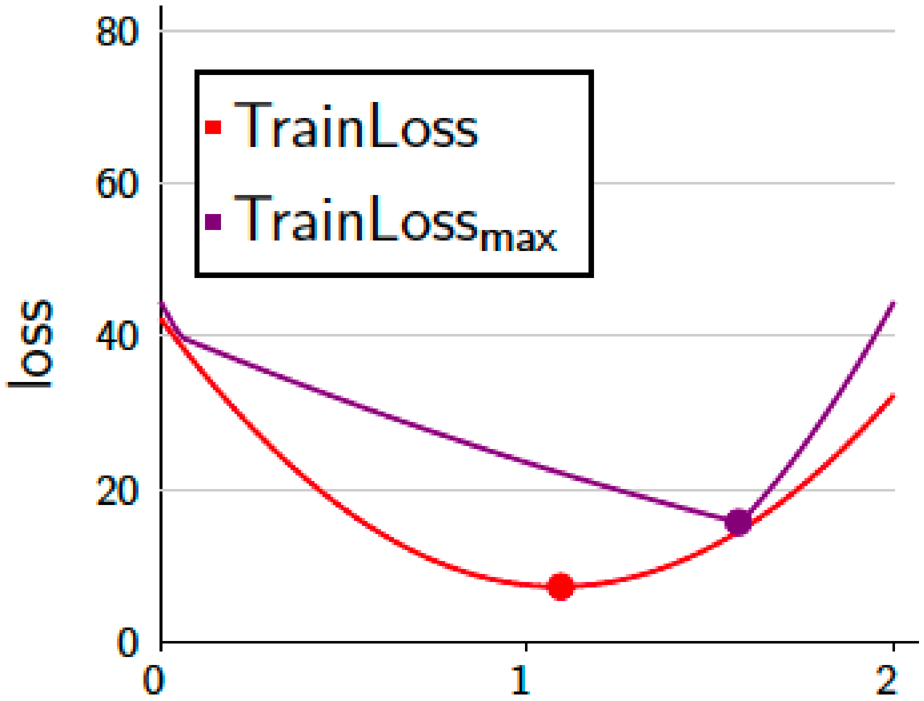
$$\min_{\beta} \max_{g} \hat{L}_g(\beta)$$

Average loss vs. maximum group loss

- Illustration in a toy example with six data points, separated into two groups

x	y	g
1	4	A
2	8	A
5	5	B
6	6	B
7	7	B
8	8	B





Training via gradient descent

- Gradient descent for group robust minimization
  - Initialize β_0
  - Let $\nabla\hat{L}(\beta_t)$ be the gradient of the training loss at β_t
  - Let η be a learning rate parameter

$$\beta_t \leftarrow \beta_t - \eta \cdot \nabla_{\mathrm{g_t}} \hat{L}\left(f_{\beta_t}\right),$$

Where $g_t \leftarrow \arg\max_{g} \hat{L}_g(\beta_t)$