Chapter 4 Classification

– Math 313 Statistics for Data Science

Guangliang Chen

Associate Professor

cheng@hope.edu

Hope College, Fall 2023

Presentation Overview

Section 4.1 An overview of classification

We have just covered in Chapter 3 the linear regression problem which deals with a continuous (or quantitative or numerical) response.

In other cases when the response is discrete (or qualitative or categorical), the task of predicting the response is called classification.

There are many classifiers that one might use to predict a categorical response. In this lecture we cover the following:

- logistic regression,
- linear discriminant analysis,
- quadratic discriminant analysis,
- naive Bayes, and
- K nearest neighbors (KNN)

Section 4.2 Why not linear regression?

When a response variable has three (or more) levels, say cat, dog, bird, simply encoding the levels by integers like 1, 2, 3 does not work well for the following reasons:

- The ordering is arbitrary and might be very wrong;
- Even when the introduced ordering might make sense, the gap between them is assumed to be equal (not necessarily true).

In the case of a binary qualitative response (with two levels), we can use a 0/1 valued dummy variable to convert it to quantitative. We can then attempt to perform regression with the 0/1 coding of the response, but need to be cautious about the interpretation.

Section 4.3 Logistic regression

The regression approach to binary classification

We just said that in the case of binary response, classification can be approached as a regression problem with 0/1 coding:

$$\underbrace{y}_{\text{binary response}} = f\left(\underbrace{x_1, \ldots, x_d}_{\text{predictors}}\right).$$

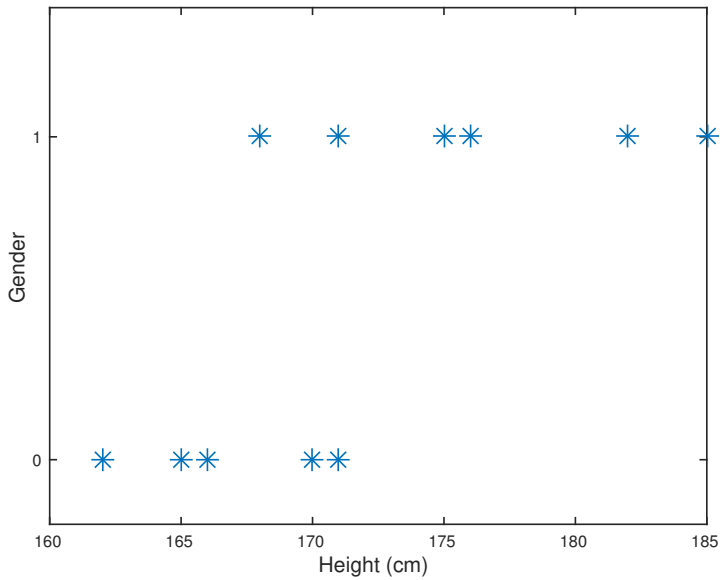
where $f: \mathbb{R}^d \mapsto \{0, 1\}$.

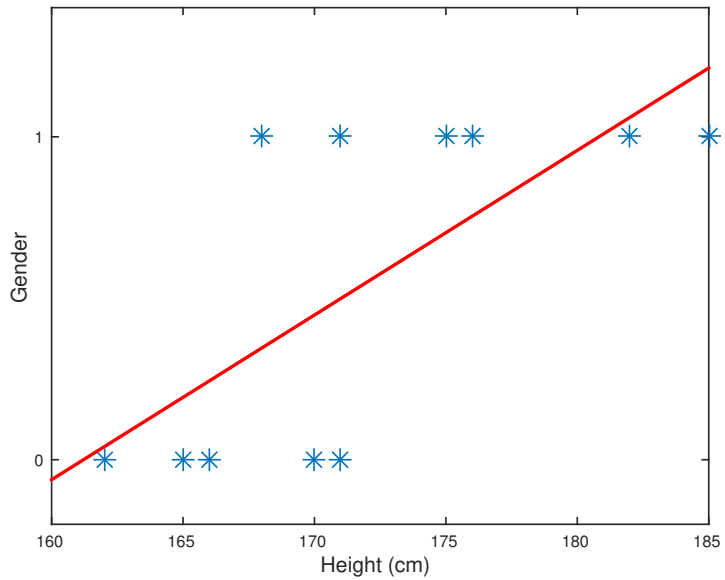
To explain the ideas, we start with a binary classification problem with only one predictor:

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

Motivating example

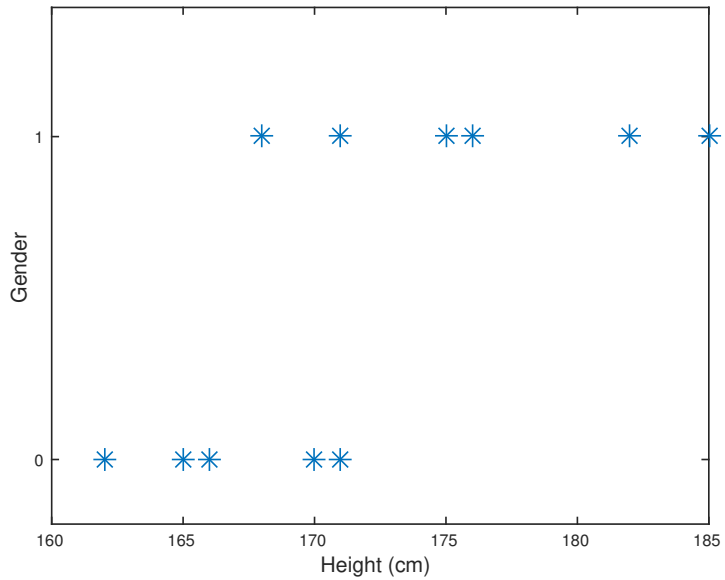
Consider a specific example where x represents a person's height while y denotes the person's sex (0 = Female, 1 = Male).

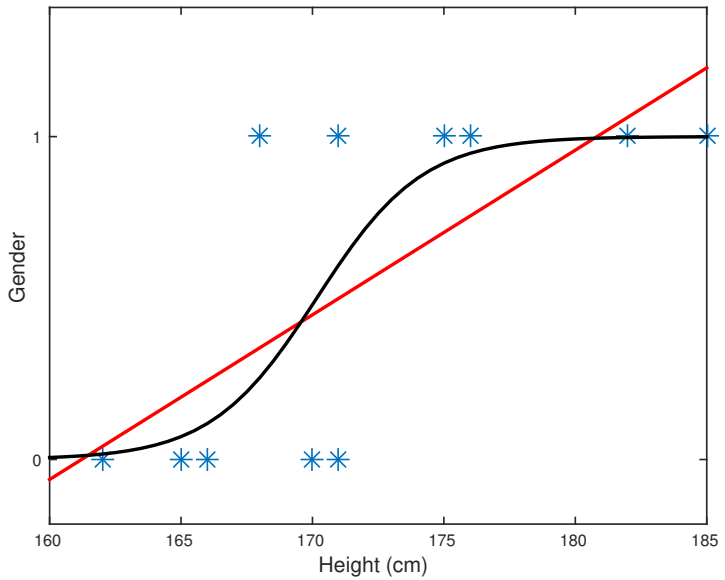




Simple linear regression is not appropriate in this case.

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



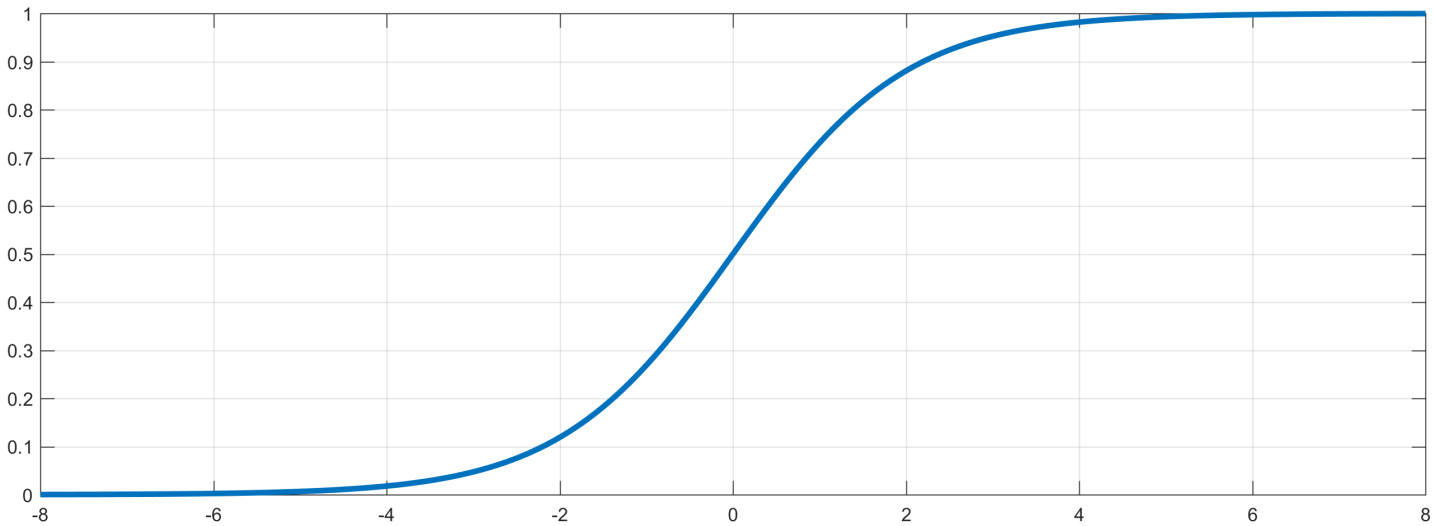


A better option is to use an S-shaped curve to fit the data.

Which functions have such shapes?

An example of such a curve is the **sigmoid** function:

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



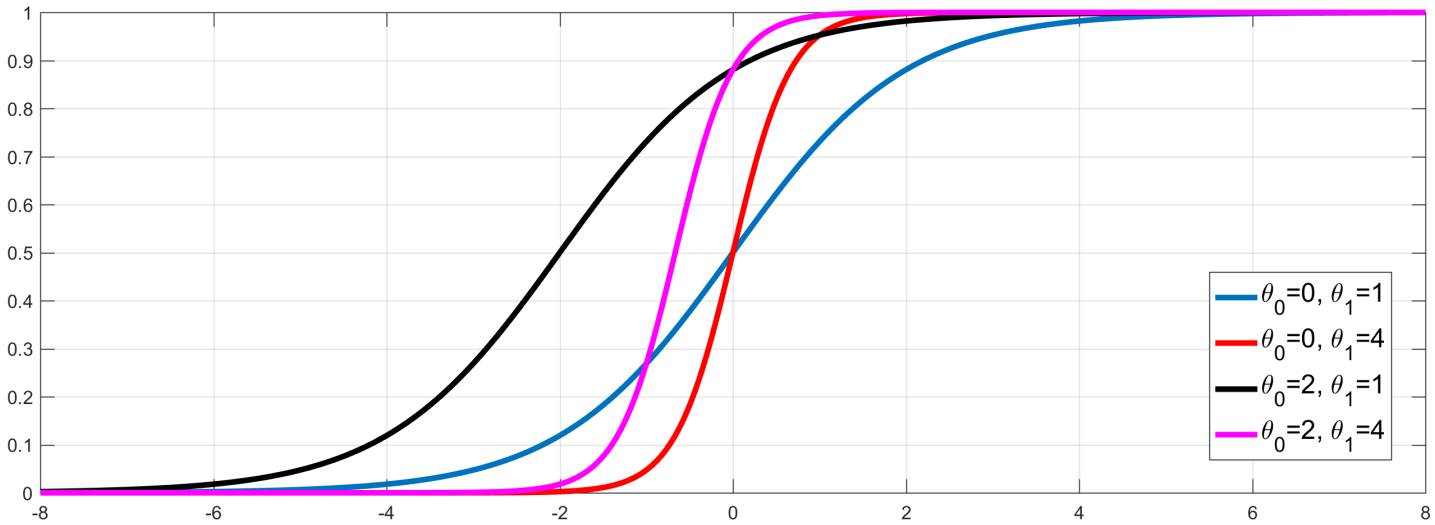
Can you think of another function that has a similar shape?

Making the logistic function more flexible

We generalize the basic sigmoid function to a **location-scale family** of similar curves (to provide abundant flexibility):

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

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



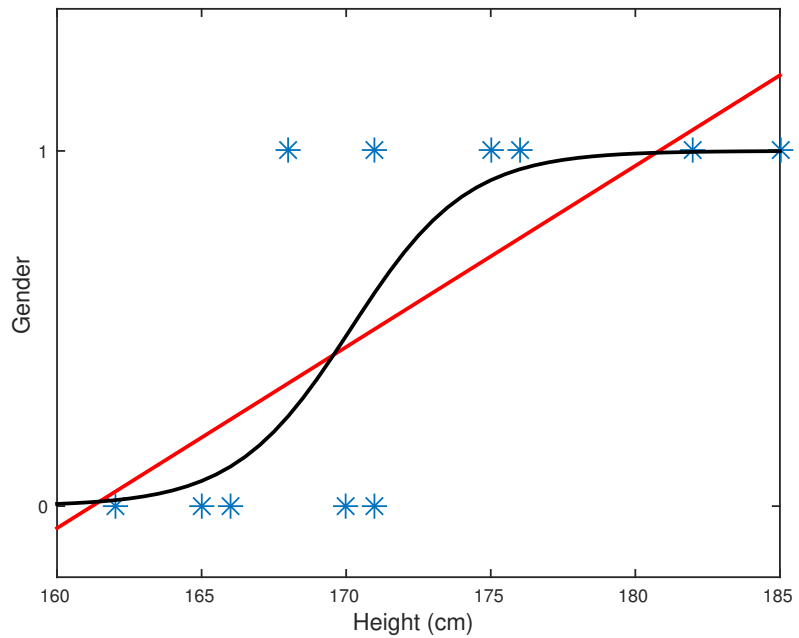
The logistic regression problem

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

Problem. Given training data $\{(x_i, y_i) \mid 1 \le i \le n\}$, with $y_i = 0, 1$, 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 "as closely as possible".



How to define the optimality

One naive choice is to use the least squares criterion based on the square loss $\ell(y, \hat{y}) = (y - \hat{y})^2$:

$$\min_{\theta_0, \theta_1} \sum_{i=1}^{n} \underbrace{(y_i - \hat{y}_i)^2}_{\text{square loss}}, \qquad \text{where} \quad \hat{y}_i = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$

It does not work well in this binary setting due to two reasons:

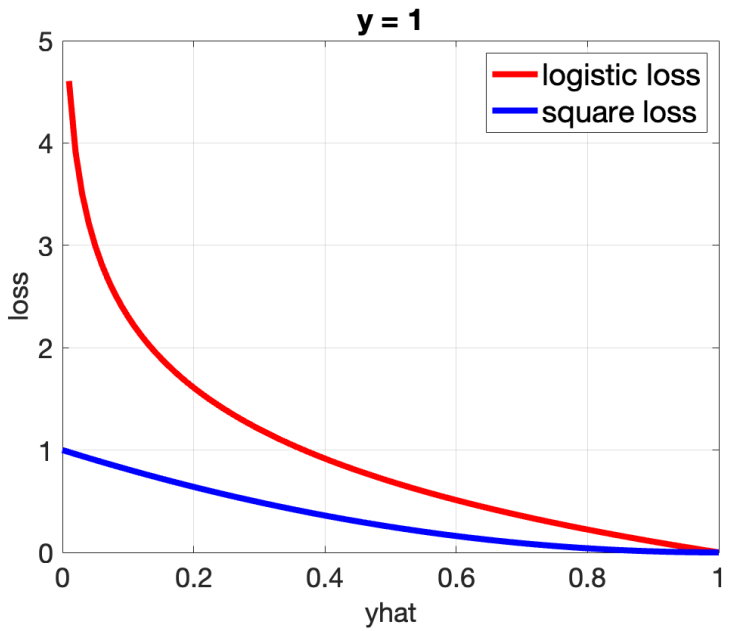
- The penalty of being totally wrong is not big enough (only 1);
- It is known to suffer from a learning slowdown when solving the optimization problem numerically through gradient descent.

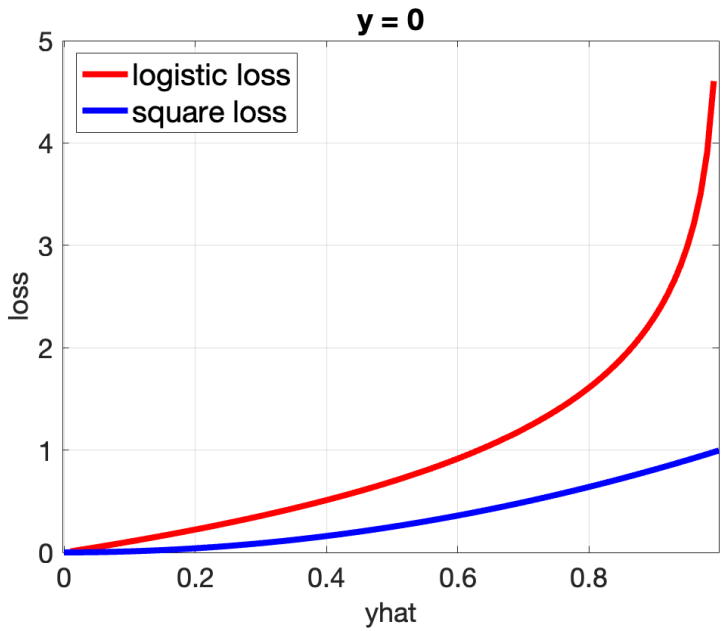
There is a better choice of loss function, called the logistic loss or the cross-entropy loss, which we introduce next.

The logistic loss function

The logistic loss is defined as follows:

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





It poses much heavier penalties on very wrong predictions than the square loss. It also has a very meaningful statistical interpretation.

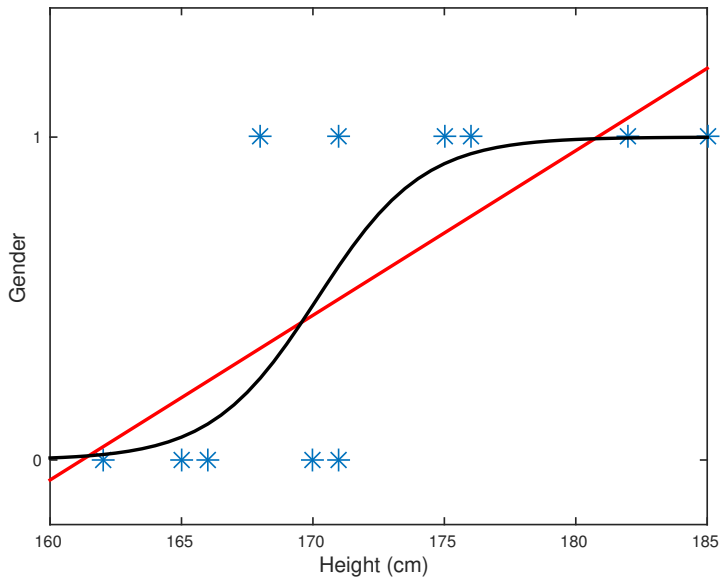
Statistical modeling of logistic regression

Key idea: To model the response Y as a Bernoulli random variable with a probability of success dependent on location x and parametrized by $\boldsymbol{\theta} = (\theta_0, \theta_1)$ (in the form of the sigmoid function):

$$Y \mid x, \boldsymbol{\theta} \sim \text{Bernoulli}(p), \quad \text{where} \quad p(x, \boldsymbol{\theta}) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$

That is,

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



Note that the pmf of Y when given x and $\boldsymbol{\theta}$ can be written as

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

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

$$L(\theta \mid (x_i, y_i), 1 \leq i \leq n) = \prod_{i=1}^{n} P(Y = y_i \mid x_i, \theta) = \prod_{i=1}^{n} p_i^{y_i}(1 - p_i)^{1-y_i}$$

where

$$p_i = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_i)}}, \quad 1 \leq i \leq n$$

The log likelihood is

$$\log L(\theta) = \sum_{i=1}^{n} (y_i \log p_i + (1 - y_i) \log(1 - p_i))$$

Its maximizer, called the Maximum Likelihood Estimator (MLE) of θ and denoted $\hat{\theta}$, gives the optimal parameter values for the model.

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

Connection to optimization

Mathematically, the MLE formulation

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

is equivalent to the minimization problem

$$\min_{\theta} -\log L(\theta) = \sum_{i=1}^{n} (-y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i))$$

The terms in the above summation for $-\log L(\theta)$, i.e.,

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

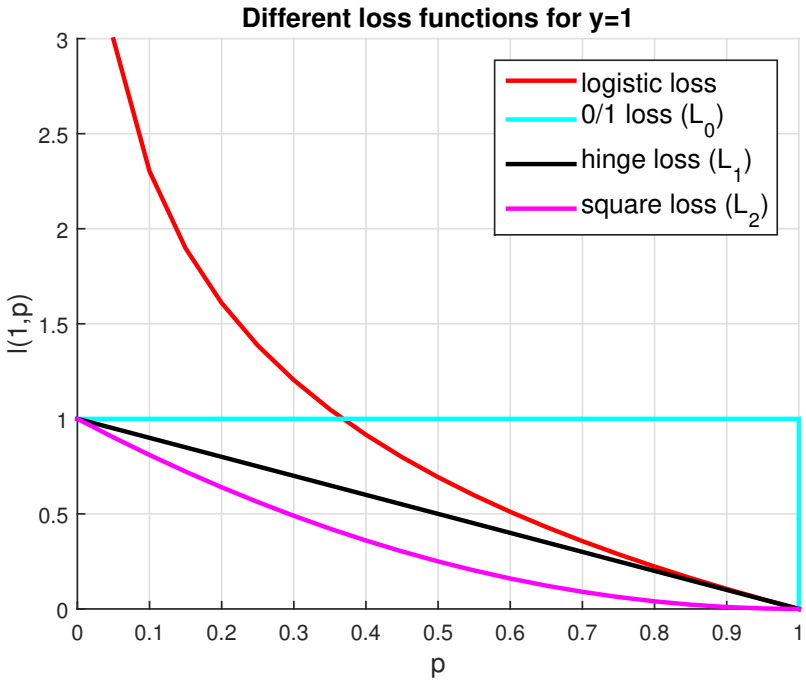
can be thought of as measuring the error between y_i and its approximation $\hat{y}_i$ (the logistic loss).

More loss functions

It turns out that by changing the choice of the loss function, we can invent more classifiers by the optimization approach.

Here are a few loss functions (with their graphs on the right):

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



Other commonly used loss functions include the log loss and exponential loss.

How to classify new observations

After we fit the logistic model to the training set (and obtain the MLE θ),

$$\hat{y} = \frac{1}{1 + e^{-(\hat{\theta}_0 + \hat{\theta}_1 x)}}$$

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

$$\text{Assign label } y = 1 \quad \text{if and only if} \quad \hat{y} > \frac{1}{2}.$$

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

$$\hat{y} > \frac{1}{2} \quad \text{if and only if} \quad \underbrace{P(Y = 1 \mid x, \theta)}_{\hat{y}} > \underbrace{P(Y = 0 \mid x, \theta)}_{1-\hat{y}}$$

The general binary classification problem

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

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

where $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_d)$ and $\mathbf{x} = (1, x_1, \ldots, x_d)$, and use the log-likelihood function to find the MLE $\hat{\boldsymbol{\theta}}$:

$$\hat{y} = \frac{1}{1 + e^{-\hat{\boldsymbol{\theta}} \cdot \mathbf{x}}}$$

The classification rule for a test point $\mathbf{x}$ remains the same:

$$y = 1_{\hat{y} > 0.5}$$

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

$$\hat{y} = \frac{1}{1 + e^{-\hat{\theta} \cdot \mathbf{x}}} = \frac{1}{2}$$

or equivalently,

$$\hat{\theta} \cdot \mathbf{x} = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \cdots + \hat{\theta}_d x_d = 0$$

which is a hyperplane in $\mathbb{R}^d$.

This shows that LR is a linear classifier.

LR is a generalized linear model (GLM)

The LR model

$$p(\mathbf{x}, \theta) = \hat{y} = \frac{1}{1 + e^{-\theta \cdot \mathbf{x}}}$$

can be rewritten as

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

where the response Y is a Bernoulli random variable with mean

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

Interpretation:

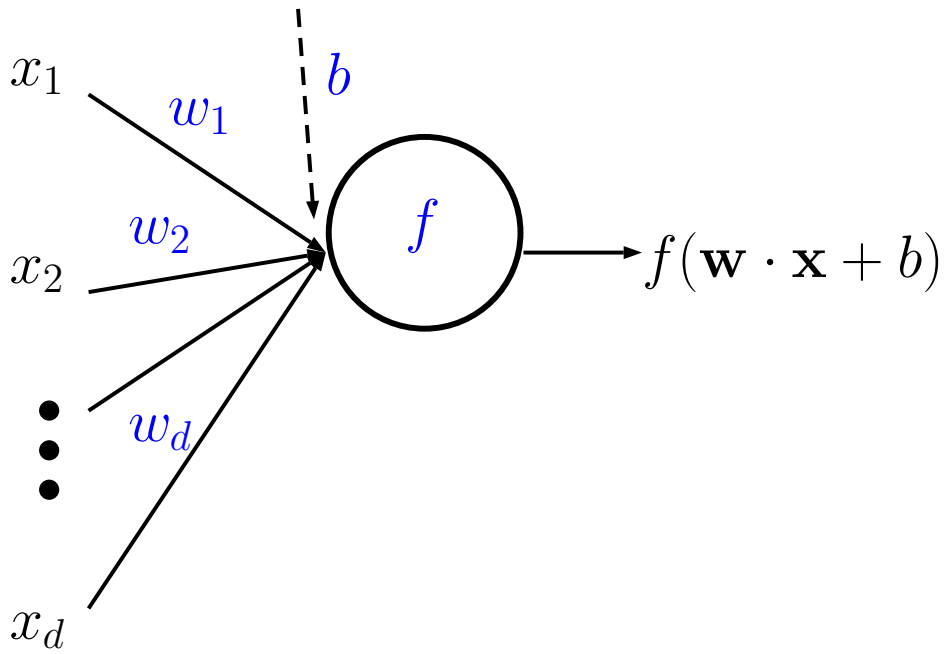
- p: probability of "success" (i.e. $Y = 1$)
- $\frac{p}{1-p}$: odds of "winning"
- $\log \frac{p}{1-p}$: logit (a link function)

LR is a single-neuron network

In logistic regression, we already see a lot of the concepts that are used in the design and training of a neural network, such as

- Activation function (sigmoid, and more)
- Loss function (square, logistic)
- Gradient descent (for solving the optimization problem)

To learn more on this, see Nielson's online book on neural networks and deep learning[a] (very easy and enjoyable to read).

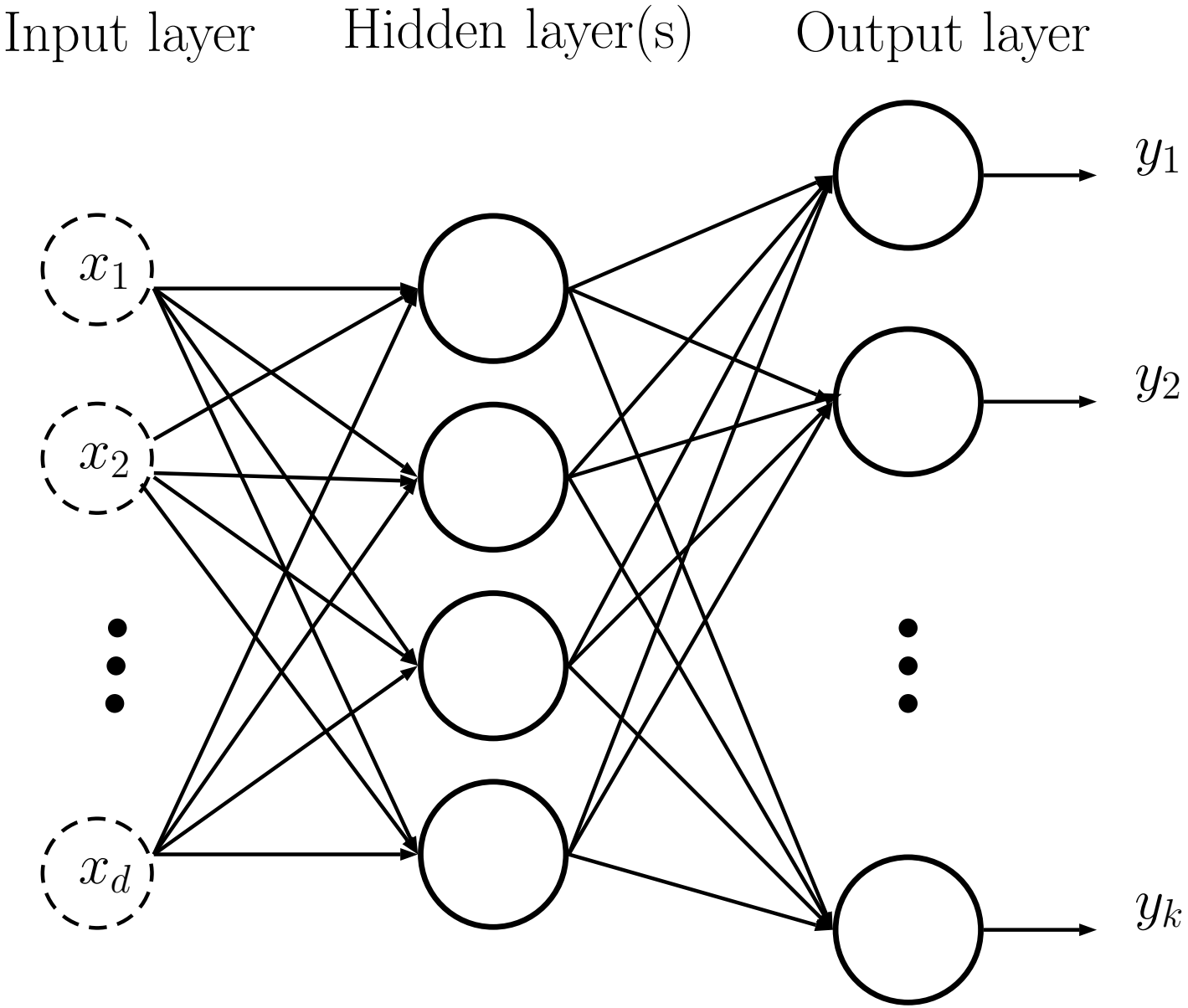


In the above,

- w_i: **weights**, b: **bias**, and f: **activation function**

[a] https://neuralnetworksanddeeplearning.com/

A neural network

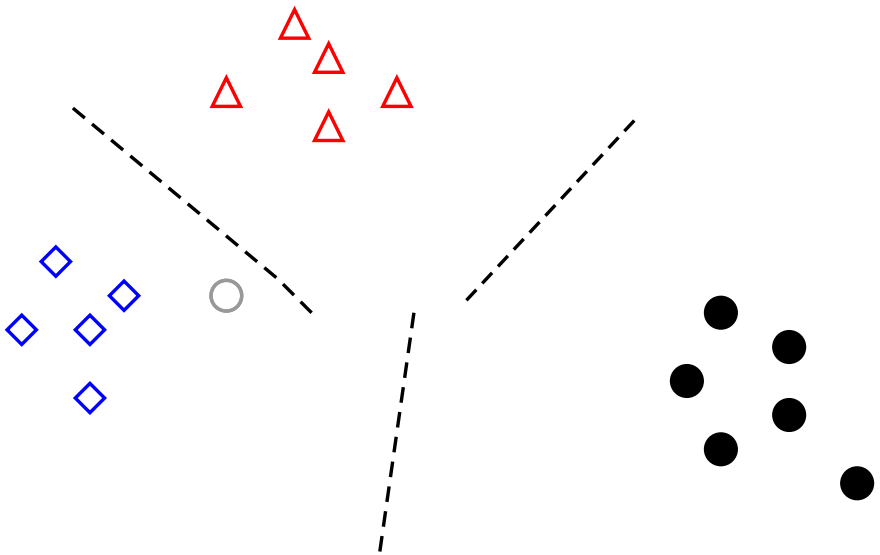


Multiclass extensions of LR

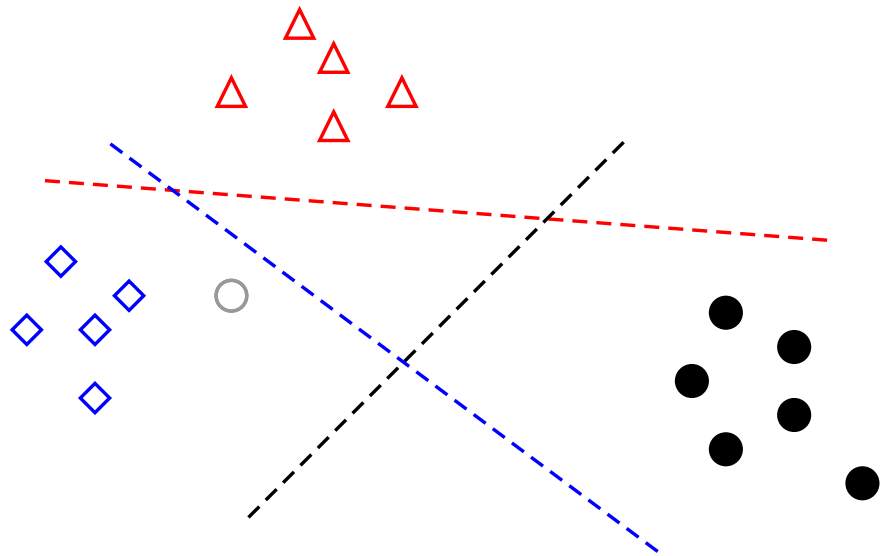
1. **Union of binary models**
   - *One versus one*: construct a LR model for every pair of classes
   - *One versus rest*: construct a LR model for each class against the rest of the training set

   In either case, the prediction of the label of a test point is collectively determined by all the models (see next slide).

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,
  - Use each class in turn as a reference class exactly once
  - Each time the chosen reference class (say class j) receives new label 1 (and the rest are given label 0). Fix a binary LR model with parameters $\hat{\boldsymbol{\theta}}^{(j)}$.
  - For each binary model with class j as the reference class, compute the probability

$$\hat{y}_0^{(j)} = P(Y_0 = 1 \mid \mathbf{x}_0, \hat{\boldsymbol{\theta}}^{(j)}) = \frac{1}{1 + e^{-\hat{\boldsymbol{\theta}}^{(j)} \cdot \mathbf{x}_0}}$$

  - The final prediction is

$$\hat{y}_0 = \arg\max_j \ \hat{y}_0^{(j)}$$

2. **Softmax regression**. It is based on the so-called **softmax function**:

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

The softmax function is a smooth function trying to approximate the indicator function (whether α_j is the largest of all):

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

See
`https://neuralnetworksanddeeplearning.com/chap3.html`.

The softmax regression fits the following model to the data:

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

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

The distribution of Y is multinomial with probabilities $P(Y = j \mid \mathbf{x}; \Theta), 1 \le j \le c$. Therefore, softmax regression is also called **multinomial logistic regression**.

Remark. Multinomial LR can be thought of as a "fixed vs rest" extension of the binary LR classifier.

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

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:

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

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

$$\begin{aligned} \hat{y}_0 &= \operatorname{argmax}_{1 \le j \le c} P(Y = j \mid \mathbf{x}_0; \widehat{\Theta}) \\ &= \operatorname{argmax}_j \frac{e^{\hat{\boldsymbol{\theta}}^{(j)} \cdot \mathbf{x}_0}}{\sum_{1 \le \ell \le c} e^{\hat{\boldsymbol{\theta}}^{(\ell)} \cdot \mathbf{x}_0}} \\ &= \operatorname{argmax}_j \hat{\boldsymbol{\theta}}^{(j)} \cdot \mathbf{x}_0 \end{aligned}$$

Feature extraction/selection for logistic regression

Logistic regression tends to overfit the data in the setting of high dimensional data (i.e., many predictors). Two ways to fix it:

- Use a **dimensionality reduction** method (such as PCA) to project data into low dimensions
- Add a **penalty** term to the objective function of the binary logistic regression classifier

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(-y_i \log \hat{y}_i - (1-y_i)\log(1-\hat{y}_i)\right) + \lambda \cdot \text{penalty}$$

where the penalty term is normally set to

- $\|\boldsymbol{\theta}\|_2^2 = \theta_0^2 + \theta_1^2 + \cdots + \theta_d^2$ (ℓ_2 regularization) or
- $\|\boldsymbol{\theta}\|_1 = |\theta_1| + \cdots + |\theta_d|$ (ℓ_1 regularization).

The constant $\lambda \geq 0$ is called a regularization parameter; larger values of λ would lead to more zeros in $\boldsymbol{\theta}$ (simpler models).

Logistic regression in Python (via sklearn)[1]

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'.

Note that regularization is applied by default.

with default values
logreg = linear_model.LogisticRegression(penalty= 'l2', C=1.0, multi_class='auto')
penalty: may be set to None, or 'l1' (which can produce more zeros in θ than 'l2')
C $(= 1/\lambda)$: inverse of regularization strength (smaller values of C specify stronger regularization).
multi_class: may be changed to 'multinomial' (no 'ovo' option)

[1] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

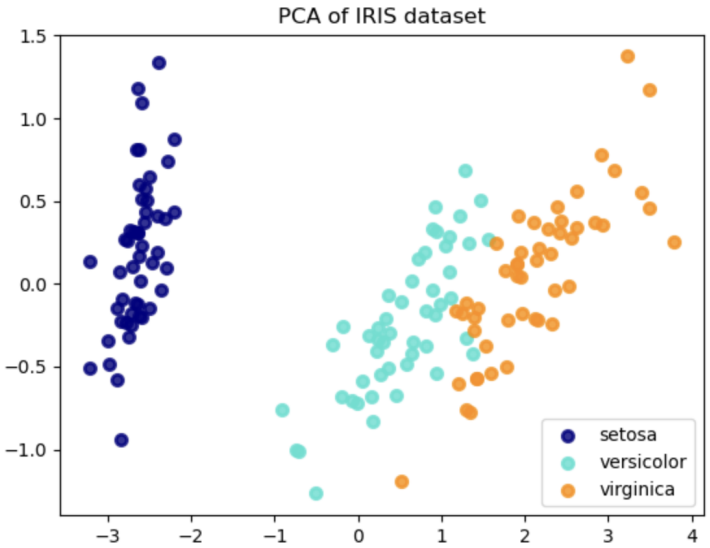
Other implementations of LR

See the Lab (Section 4.7 of the lab) and also the following:

`https://library.virginia.edu/data/articles/logistic-regression-four-ways-with-python`

Practice questions for LR

1. Textbook question 12 (page 196)
2. Consider the *iris* data used in Chapter 12 AP. First standardize the predictors and then use PCA to reduce the data to two dimensions. Now focus on the two classes, 'versicolor' and 'virginica', and fit a binary logistic regression model. What is the training error? Plot also the decision boundary.



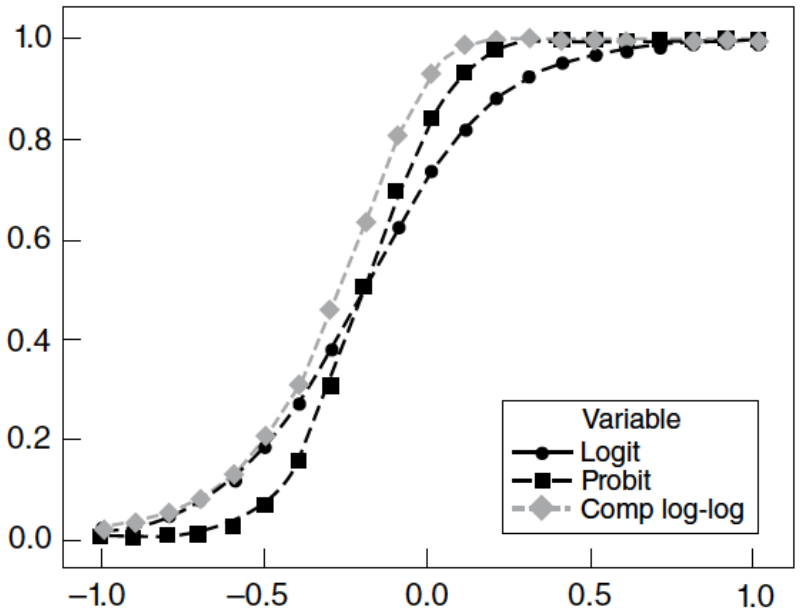
3. Show that for the sigmoid function $g(z) = \frac{1}{1+e^{-z}}$, we must have $g'(z) = g(z)(1 - g(z))$ for all $z \in \mathbb{R}$. Where is $g'(z)$ largest?

4. It was mentioned that logistic regression uses the logit link function (which leads to the sigmoid function):

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

Other choices of the link function include the following:

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



Derive a formula for p (in terms of $\mathbf{x}$ and θ) by using the Cauchit link function.

Section 4.4 Generative models for classification

Recall the statistical perspective of classification

Let $\vec{X} \in \mathbb{R}^d$, $Y \in \mathbb{R}$ be two random variables representing the location and label of a data point to be observed.

Suppose they have a joint distribution $f_{\vec{X},Y}$, and marginal distributions $f_{\vec{X}}$ (continuous) and f_Y (discrete).

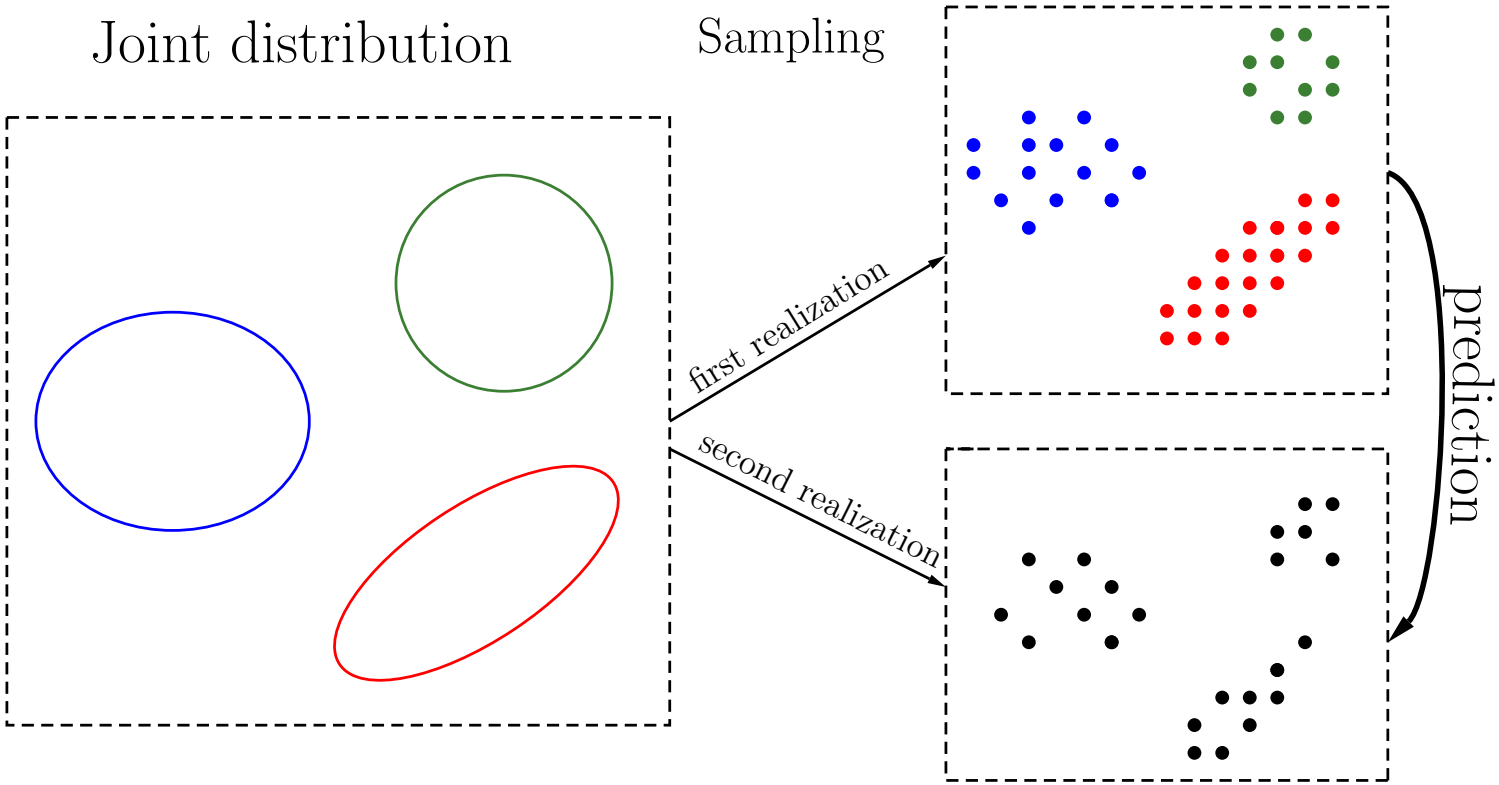
The training data can be modeled by a random sample from the joint distribution:

$$(\vec{X}_1, Y_1), \ldots, (\vec{X}_n, Y_n) \overset{iid}{\sim} f_{\vec{X},Y}$$

The test data (without labels) is an independent sample from the marginal distribution of $\vec{X}$:

$$\vec{X}_{n+1}, \ldots, \vec{X}_{n+m} \overset{iid}{\sim} f_{\vec{X}}$$

The problem of classification is thus to predict the value of the label for each of the test point locations $\vec{X}_{n+j},\ 1 \leq j \leq m$.



Prior probabilities

Let the range of Y be $\{1, 2, \ldots, c\}$, with probabilities

$$P(Y = j) = \pi_j, \quad 1 \leq j \leq c.$$

We call π_j a prior probability, i.e., the probability that a new point belongs to class j before it is seen (i.e., the value of $\vec{X}$ has not been observed).

A naive way would be to assign any new data point to the class with the largest prior probability

$$\hat{j} = \text{argmax}_j \, \pi_j$$

This method makes a constant prediction (i.e., the most frequent value of Y) with test error rate $1 - \pi_{\hat{j}}$.

We don't know the true values of π_j, so we'll estimate them using a sample.

Given a specific training data set

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n),$$

a point estimate of π_j is

$$\hat{\pi}_j = \frac{n_j}{n}, \quad \forall\, j = 1, \ldots, c$$

where n_j is the actual number of training points from C_j:

$$n_j = \sum_{i=1}^{n} 1_{y_i = j}, \quad 1 \leq j \leq c$$

Bayes classification rule

Given a new data point **x**, a better way is to assign the label based on the largest posterior probability:

$$\hat{j} = \text{argmax}_j \, P(Y = j \mid \mathbf{x}) \longleftarrow \text{generic Bayes classifier}$$

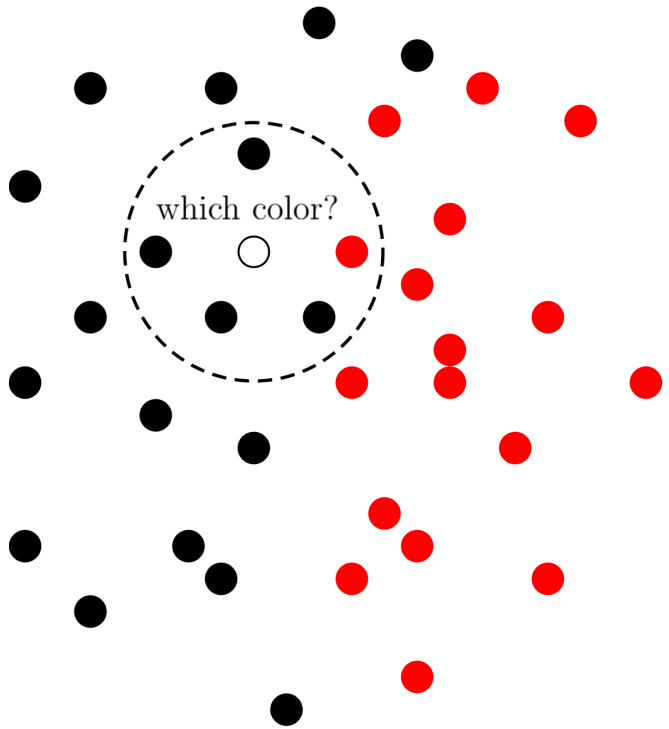
The Bayes classifier is optimal with **Bayes error rate**

$$1 - E_{\vec{X}} \left(\max_{1 \le j \le c} P(Y = j \mid \vec{X}) \right)$$

This is the lowest possible test error rate that may be achieved by a classifier.

Remark. Recall that *k*NN is also a Bayes classifier (but it is nonparametric):

$$P(Y = j \mid \mathbf{x}) \approx \frac{\#\text{nearest neighbors from class } j}{\#\text{all nearest neighbors examined}}$$



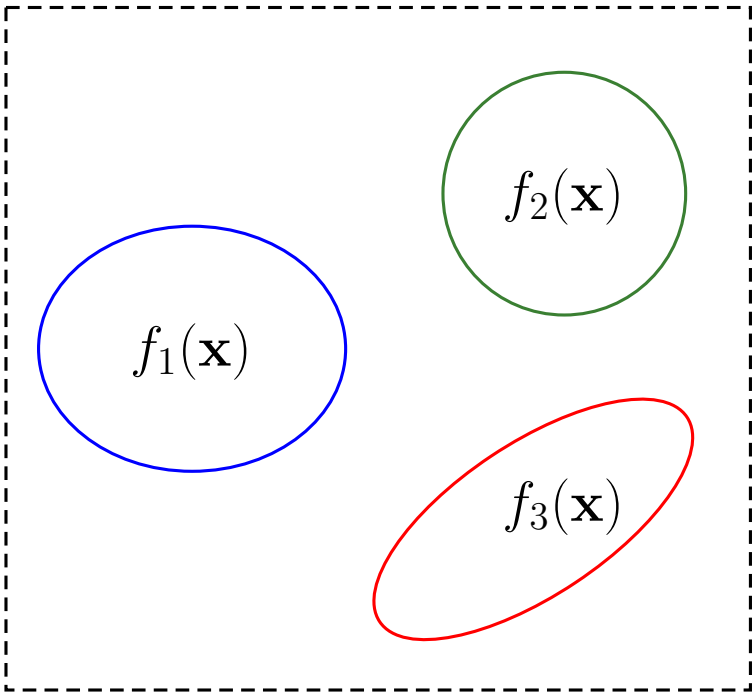
Model-based Bayes classification

Suppose that for each $j = 1, \ldots, c$,

$$P(Y = j) = \pi_j$$
$$f(\mathbf{x} \mid Y = j) = f_j(\mathbf{x}).$$

The marginal density of $\vec{X}$ is

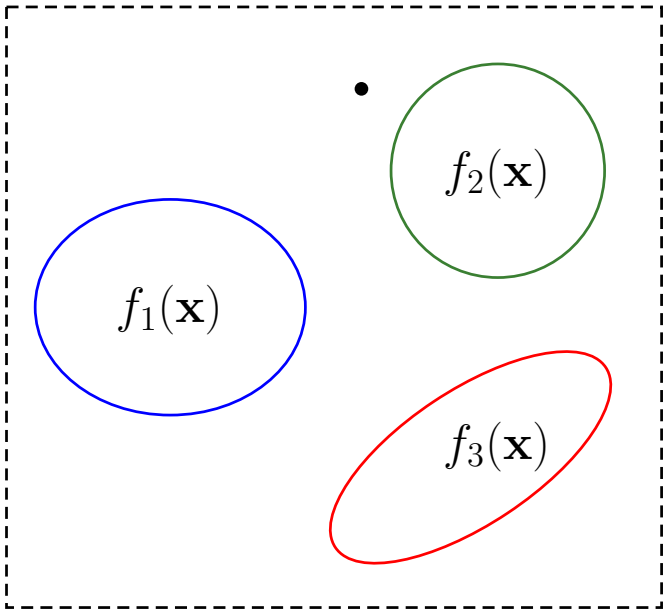
$$\begin{aligned} f_{\vec{X}}(\mathbf{x}) &= \sum_j f_{\vec{X},Y}(\mathbf{x}, Y = j) \\ &= \sum_j f(\mathbf{x} \mid Y = j)\, P(Y = j) \\ &= \sum_j f_j(\mathbf{x})\pi_j \end{aligned}$$



According to Bayes' rule, the posterior probabilities are given by

$$P(Y = j \mid \mathbf{x}) = \frac{f(\mathbf{x} \mid Y = j) \cdot P(Y = j)}{f(\mathbf{x})}$$
$$\propto f_j(\mathbf{x})\, \pi_j$$

where $j = 1, \ldots, c$.



Therefore, the Bayes classification rule can be stated as

$$\hat{j} = \text{argmax}_j \underbrace{f_j(\mathbf{x})}_{\text{likelihood}} \cdot \underbrace{\pi_j}_{\text{prior prob}}$$

$\longleftarrow$ model-based Bayes classifier

Estimating class-conditional probabilities $f_j(\mathbf{x})$

We need to select a template (i.e., model) for each component first.

Different choices of $f_j(\mathbf{x})$ lead to different Bayes classifiers:

- **LDA/QDA** - multivariate Gaussian distributions

$$f_j(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}_j|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T \mathbf{\Sigma}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)}, \quad j = 1, \ldots, c$$

- **Naive Bayes** - by assuming independent features in $\mathbf{x} = (x_1, \ldots, x_d)$:

$$f_j(\mathbf{x}) = \prod_{k=1}^{d} f_{jk}(x_k) \longleftarrow \text{1D distributions to be specified}$$

What are multivariate Gaussians?

Briefly speaking, they are generalizations of the 1D Gaussian distribution

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

to higher dimensions:

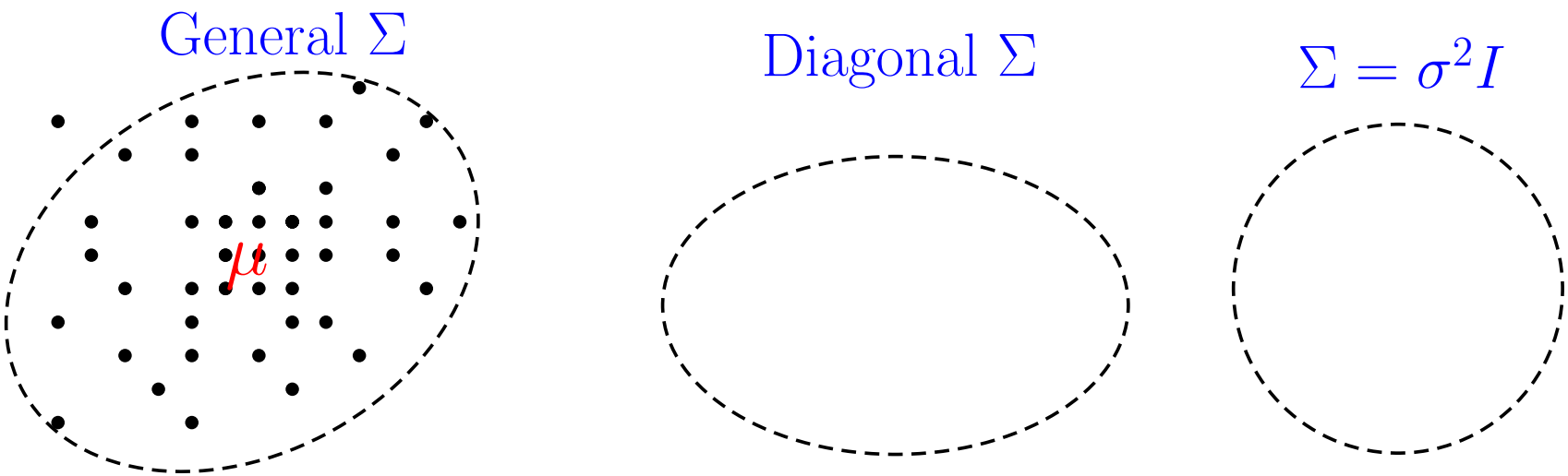
$$f(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}, \quad \mathbf{x} \in \mathbb{R}^d$$

Remark. If $\mathbf{\Sigma} = \sigma^2\mathbf{I}$ (i.e., constant diagonal), then the above formula reduces to

$$f(\mathbf{x}) = \frac{1}{(2\pi\sigma^2)^{d/2}} e^{-\frac{\|\mathbf{x}-\boldsymbol{\mu}\|^2}{2\sigma^2}}, \quad \mathbf{x} \in \mathbb{R}^d$$

In the pdf of a multivariate Gaussian,

- $\boldsymbol{\mu} \in \mathbb{R}^d$: center of the distribution
- $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$: covariance matrix



The Bivariate normal ($d = 2$)

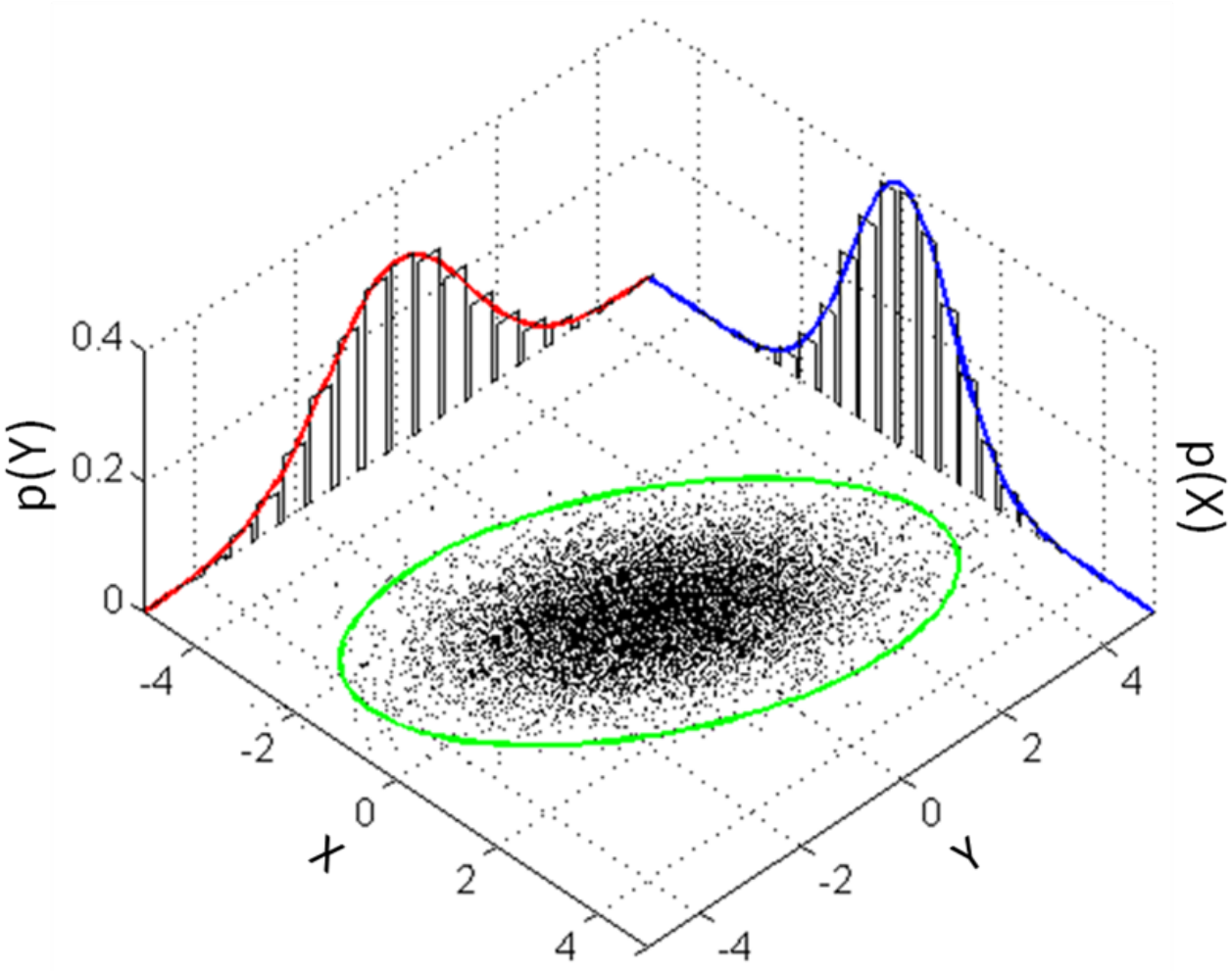
Write

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$$

The joint density is

$$f(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \cdot \exp\left(-\frac{1}{2(1-\rho^2)} \left[\frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} \right] \right)$$

Marginals of the bivariate normal are 1D normal distributions: $N(\mu_i, \sigma_i^2)$



Bayes classification with multivariate Gaussians

Under such a mixture of Gaussians model,

$$f_j(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}_j|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T\mathbf{\Sigma}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)}, \quad \forall\, j = 1, \ldots, c$$

the Bayes classification rule (for a new data point $\mathbf{x}$)

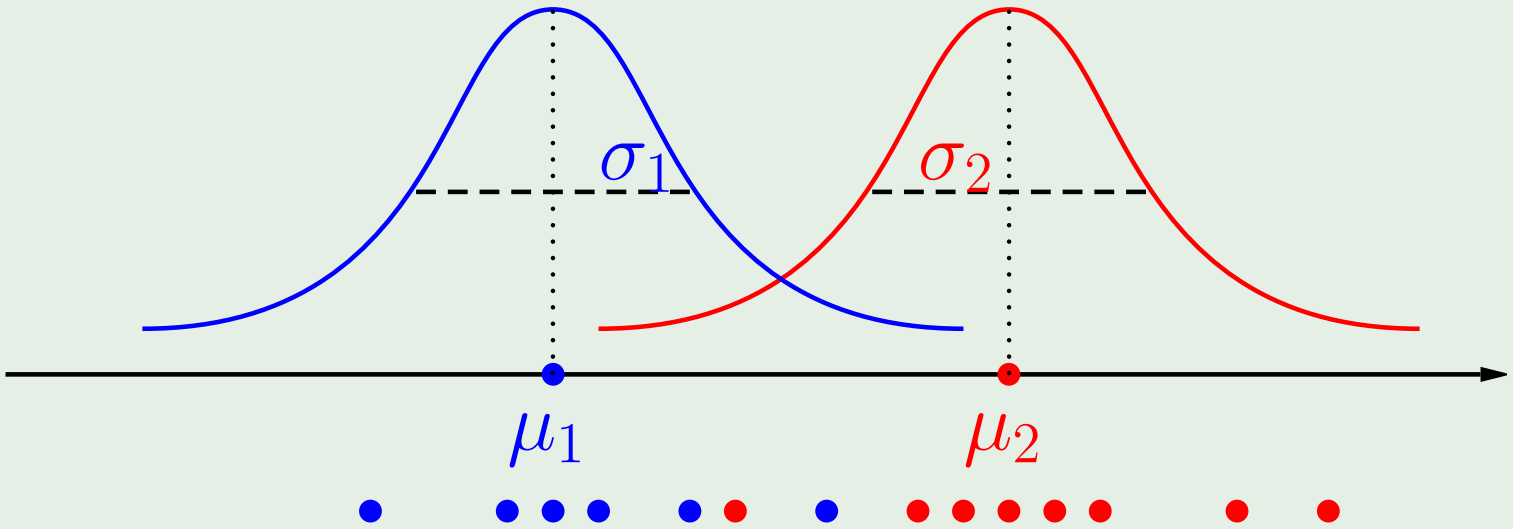
$$\hat{j} = \text{argmax}_j \; f_j(\mathbf{x})\pi_j$$

becomes the following:

$$\begin{aligned}\hat{j} &= \text{argmax}_j \; \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}_j|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T\mathbf{\Sigma}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)} \cdot \pi_j \\ &= \text{argmax}_j \; \log \pi_j - \frac{1}{2}\log|\mathbf{\Sigma}_j| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T\mathbf{\Sigma}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\end{aligned}$$

Example

Let's consider the special case of two 1D Gaussians:



Suppose we know the true values of $\mu_1, \mu_2, \sigma_1, \sigma_2$. The corresponding Bayes decision rule is

$$\hat{j} = \text{argmax}_j \ \log \pi_j - \frac{1}{2}\log(\sigma_j^2) - \frac{(x-\mu_j)^2}{2\sigma_j^2}$$

Remark.

- If $\pi_1 = \pi_2$ and $\sigma_1 = \sigma_2$, then the rule will assign x to the closer mean μ_j (larger π_j will favor the class further).
- The boundary point can be found by solving the following (quadratic) equation

$$\log \pi_1 - \frac{1}{2}\log(\sigma_1^2) - \frac{(x-\mu_1)^2}{2\sigma_1^2} = \log \pi_2 - \frac{1}{2}\log(\sigma_2^2) - \frac{(x-\mu_2)^2}{2\sigma_2^2}$$

To simplify the math, we assume that the two components have equal variance (i.e., $\sigma_1 = \sigma_2 = \sigma$), in which case we obtain

$$x = \frac{\mu_1 + \mu_2}{2} + \frac{\sigma^2 \log(\pi_1/\pi_2)}{\mu_2 - \mu_1}$$

Quadratic Discriminant Analysis (QDA)

The decision boundary of a classifier consists of points that have a tie.

For the Bayes classification rule based on a mixture of Gaussians model, the decision boundaries are given by

$$\begin{aligned} &\log \pi_j - \frac{1}{2}\log|\mathbf{\Sigma}_j| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T\mathbf{\Sigma}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j) \\ =&\log \pi_\ell - \frac{1}{2}\log|\mathbf{\Sigma}_\ell| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_\ell)^T\mathbf{\Sigma}_\ell^{-1}(\mathbf{x}-\boldsymbol{\mu}_\ell) \end{aligned}$$

This shows that such a Bayes classifier has quadratic boundaries (between each pair of training classes), and is thus called *Quadratic Discriminant Analysis (QDA)*.

Parameter estimation for QDA

The QDA classifier

$$\hat{j} = \text{argmax}_j \ \log \pi_j - \frac{1}{2} \log |\mathbf{\Sigma}_j| - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^T \mathbf{\Sigma}_j^{-1}(\mathbf{x} - \boldsymbol{\mu}_j)$$

depends on the model parameters $\pi_j, \boldsymbol{\mu}_j, \mathbf{\Sigma}_j$ but their true values are typically unknown.

Given training data, we estimate them as follows:

$$\hat{\pi}_j = \frac{n_j}{n}, \quad \hat{\boldsymbol{\mu}}_j = \frac{1}{n_j} \sum_{\mathbf{x} \in C_j} \mathbf{x}, \quad \hat{\mathbf{\Sigma}}_j = \frac{1}{n_j - 1} \sum_{\mathbf{x} \in C_j} (\mathbf{x} - \hat{\boldsymbol{\mu}}_j)(\mathbf{x} - \hat{\boldsymbol{\mu}}_j)^T$$

Linear Discriminant Analysis (LDA)

In QDA we assume that the component distributions are all multivariate Gaussians but allow them to have separate means $\boldsymbol{\mu}_j$ and covariances $\boldsymbol{\Sigma}_j$.

However, these are a lot of parameters to be estimated from the training data (especially when the dimension is large).

There is also a risk of overfitting the data.

To ease the computational burden, we assume that all the components have the same covariance

$$\boldsymbol{\Sigma}_1 = \cdots = \boldsymbol{\Sigma}_c = \boldsymbol{\Sigma}$$

In this special setting, the Bayes classification rule becomes

$$\begin{aligned}\hat{j} &= \text{argmax}_j \ \log \pi_j - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_j) \\ &= \text{argmax}_j \ \mathbf{x}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_j - \frac{1}{2}\boldsymbol{\mu}_j^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_j + \log \pi_j.\end{aligned}$$

The decision boundary between two classes (j, ℓ) is

$$\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_j - \frac{1}{2}\boldsymbol{\mu}_j^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_j + \log \pi_j = \mathbf{x}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_\ell - \frac{1}{2}\boldsymbol{\mu}_\ell^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_\ell + \log \pi_\ell$$

which simplifies to

$$\mathbf{x}^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_\ell) = \frac{1}{2}\left(\boldsymbol{\mu}_j^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_j - \boldsymbol{\mu}_\ell^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_\ell\right) + \log \frac{\pi_\ell}{\pi_j}$$

This is a hyperplane with normal vector $\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_\ell)$.

Parameter estimation for LDA

Similarly, we can use the training data to estimate the parameters $\pi_j, \boldsymbol{\mu}_j$ as follows:

$$\hat{\pi}_j = \frac{n_j}{n}, \quad \hat{\boldsymbol{\mu}}_j = \frac{1}{n_j} \sum_{\mathbf{x} \in C_j} \mathbf{x}$$

For the shared covariance matrix $\boldsymbol{\Sigma}$, we use the following pooled estimator:

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{n-c} \sum_{j=1}^{c} \sum_{\mathbf{x} \in C_j} (\mathbf{x} - \hat{\boldsymbol{\mu}}_j)(\mathbf{x} - \hat{\boldsymbol{\mu}}_j)^T$$

This leads to the following LDA decision rule:

$$\hat{j} = \text{argmax}_j \, \mathbf{x}^T \hat{\boldsymbol{\Sigma}}^{-1} \hat{\boldsymbol{\mu}}_j - \frac{1}{2} \hat{\boldsymbol{\mu}}_j^T \hat{\boldsymbol{\Sigma}}^{-1} \hat{\boldsymbol{\mu}}_j + \log \hat{\pi}_j.$$

The singularity issue in LDA/QDA

Both LDA and QDA require inverting covariance matrices, which may be (nearly) singular in the case of high dimensional data.

Common fixes:

- Apply PCA to reduce dimensionality first, or
- Regularize the covariance matrices, i.e.,

$$\text{QDA}: \quad \hat{\mathbf{\Sigma}}_j \longleftarrow \hat{\mathbf{\Sigma}}_j + \beta \mathbf{I}, \; j = 1, \ldots, c$$
$$\text{LDA}: \quad \hat{\mathbf{\Sigma}} \longleftarrow \hat{\mathbf{\Sigma}} + \beta \mathbf{I}$$

where $\beta \geq 0$ is a tuning parameter.

Python scripts for LDA/QDA

See

`https://scikit-learn.org/stable/modules/classes.html#module-sklearn.discriminant_analysis`

Naive Bayes

The naive Bayes classifier is also based on the Bayes decision rule:

$$\hat{j} = \text{argmax}_j \, f_j(\mathbf{x})\pi_j$$

It assumes that the individual components of $\vec{X} = (X_1, \ldots X_d)$ are conditionally independent given a class $Y = j$:

$$f_j(\mathbf{x}) = f_{\vec{X}}(\mathbf{x} \mid Y = j) = \prod_{s=1}^{d} f_{X_s}(x_s \mid Y = j) = \prod_{s=1}^{d} f_{sj}(x_s).$$

This thus reduces the high dimensional density estimation problem ($\{f_j(\mathbf{x})\}_j$) to a union of simple 1D problems ($\{f_{sj}(x_s)\}_{j,s}$).

Depending on the data types and the corresponding models used for the component density functions $\{f_{sj}(x_s)\}_{j,s}$, the naive Bayes classifier has different versions:

- Continuous data: Gaussian naive Bayes
- Binary/Boolean data: Bernoulli naive Bayes
- Frequency/count data: Multinomial naive Bayes
- Categorical/discrete data: multivariate multinomial naive Bayes

Gaussian naive Bayes

For continuous features (e.g., image data), the standard choice for modeling $\{f_{sj}(x_s)\}_{j,s}$ is the 1D normal distribution:

$$f_{sj}(x_s) = \frac{1}{\sqrt{2\pi}\sigma_{sj}} e^{-(x_s-\mu_{sj})^2/2\sigma_{sj}^2}$$

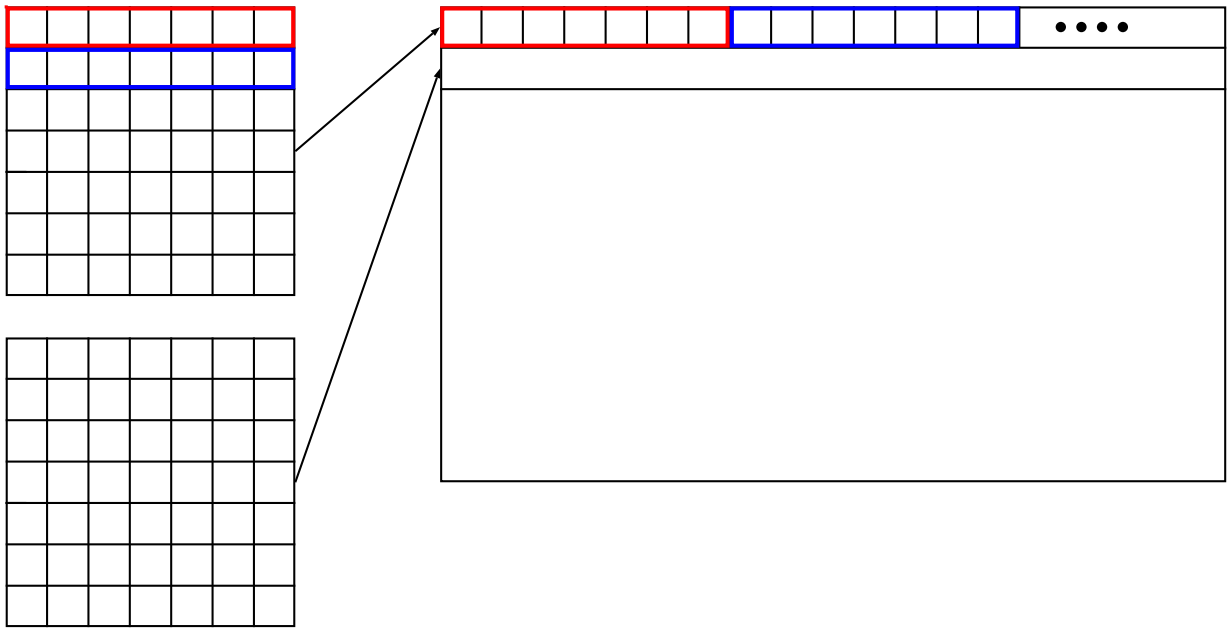
where μ_{sj}, σ_{sj} can be estimated similarly using the training data (in every class j and in every dimension s).

The resulting classification rule for a new data point $\mathbf{x} = (x_1, \ldots, x_d)'$ is

$$\hat{j} = \text{argmax}_j \ \log \hat{\pi}_j - \sum_{s=1}^{d} \left[\log \hat{\sigma}_{sj} + (x_s - \hat{\mu}_{sj})^2/2\hat{\sigma}_{sj}^2 \right]$$

Gaussian naive Bayes can be used to classify digital images (such as the MNIST handwritten digits).

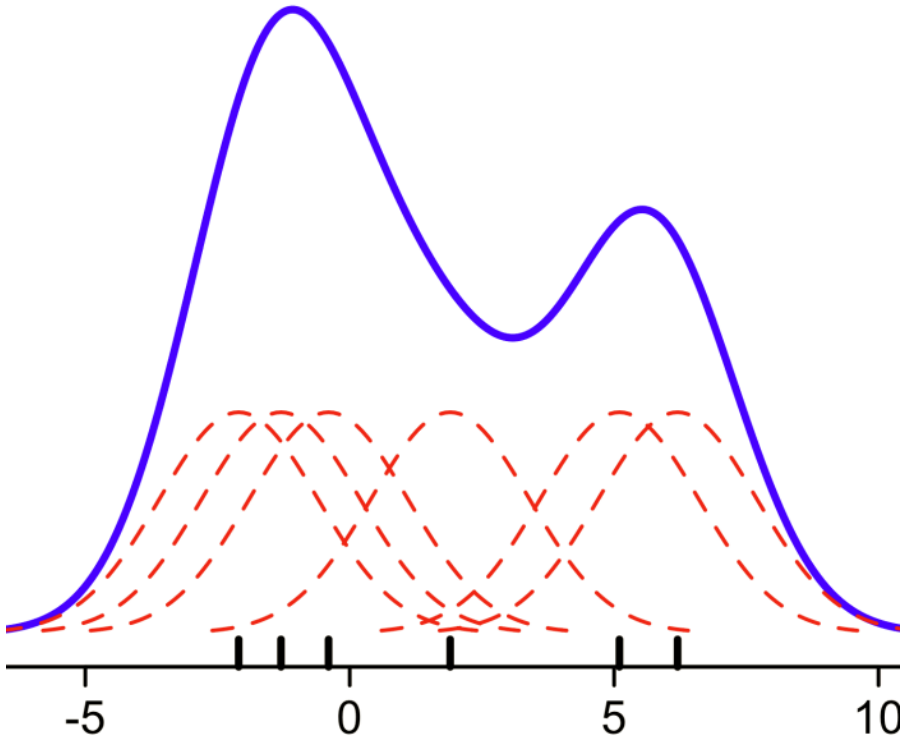
In this case, each pixel is a feature, and their intensities are assumed to be independent random variables.



Improving Gaussian naive Bayes

1. **Independence assumption**: Apply PCA to obtain uncorrelated features (closer to being independent) and meanwhile get rid of low-variance dimensions

2. **Choice of distribution**: Use kernel smoothing instead of Gaussian to better model feature distributions
However, this will be at the expense of speed.



Bernoulli naive Bayes

For binary features (i.e., 0/1 valued), we can use the Bernoulli distribution to modeling them:

$$f_{sj}(x_s) = p_{sj}^{x_s}(1 - p_{sj})^{1-x_s}, \quad x_s = 0, 1$$

where p_{sj} is the probability of X_s taking the value of 1 within class j. Similarly, it can be estimated using the training data.

The resulting classification rule for a new data point $\mathbf{x} = (x_1, \ldots, x_d)'$ is

$$\hat{j} = \text{argmax}_j \ \log \hat{\pi}_j + \sum_{s=1}^{d} \left[x_s \log \hat{p}_{sj} + (1 - x_s) \log(1 - \hat{p}_{sj}) \right]$$

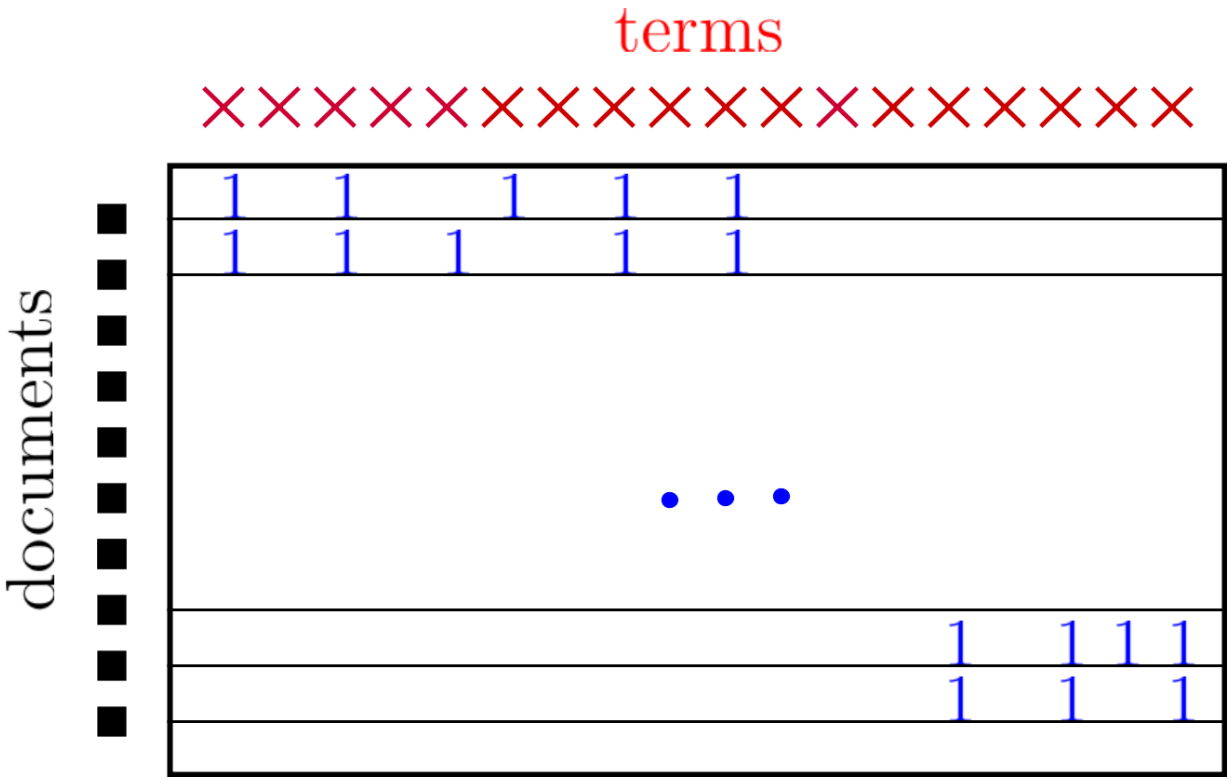
Remark. Bernoulli naive Bayes is a linear classifier, because for each j,

$$\begin{aligned}
&\log \hat{\pi}_j + \sum_{s=1}^{d} \left[x_s \log \hat{p}_{sj} + (1 - x_s) \log(1 - \hat{p}_{sj}) \right] \\
=&\log \hat{\pi}_j + \sum_{s=1}^{d} \left[x_s \log \frac{\hat{p}_{sj}}{1 - \hat{p}_{sj}} + \log(1 - \hat{p}_{sj}) \right] \\
=&\mathbf{w}_j^T \cdot \mathbf{x} + b_j
\end{aligned}$$

where

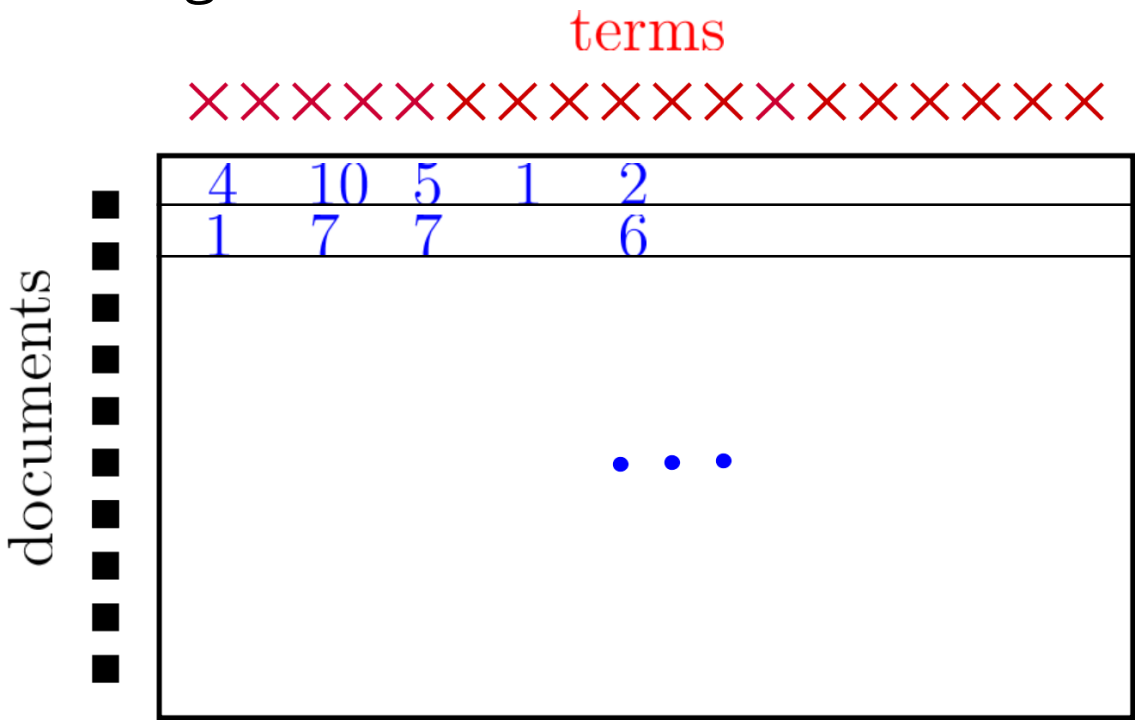
$$\mathbf{w}_j = \left(\log \frac{\hat{p}_{1j}}{1 - \hat{p}_{1j}}, \ldots, \log \frac{\hat{p}_{dj}}{1 - \hat{p}_{dj}} \right)^T, \quad b_j = \log \hat{\pi}_j + \sum_{s=1}^{d} \log(1 - \hat{p}_{sj})$$

Remark. Bernoulli naive Bayes is popular for document classification tasks where binary term occurrence features are used (1 if the term occurs in the document, 0 otherwise).



Multinomial naive Bayes

This Bayes classifier is very useful for modeling count data, such as the bag-of-words model for text documents.



To classify text documents, multinomial naive Bayes assumes that each document is a collection of frequency counts

$$\vec{X} = (X_1, X_2, \ldots, X_d)$$

of terms that are randomly and independently selected from a vocabulary of size d according to a multinomial distribution at the class level:

$$f_j(\mathbf{x}) = f_{\vec{X}|Y}(\mathbf{x} \mid Y = j) = \frac{(\sum x_s)!}{\prod x_s!} \prod_{s=1}^{d} p_{sj}^{x_s}, \quad \text{for all } x_1, \ldots, x_s \geq 0$$

where j is fixed and

$$p_{sj} \geq 0, \text{ for each } 1 \leq s \leq d, \qquad \text{and} \quad \sum p_{sj} = 1$$

are the probabilities of generating terms from a fixed topic.

It can be shown that the MLE of p_{sj} based on the jth training class is

$$\hat{p}_{sj} = \frac{\sum_{i=1}^{n_j} x_s^{(i)}}{\sum_{s=1}^{d}\sum_{i=1}^{n_j} x_s^{(i)}} \xrightarrow[\text{smoothing}]{\text{Laplace}} \frac{1+\sum_{i=1}^{n_j} x_s^{(i)}}{d+\sum_{s=1}^{d}\sum_{i=1}^{n_j} x_s^{(i)}}$$

where $x_s^{(i)}$ is the (observed) frequency of word s in the ith document of class j and

- $\sum_{i=1}^{n_j} x_s^{(i)}$: word count of sth term over all documents in class j
- $\sum_{s=1}^{d}\sum_{i=1}^{n_j} x_s^{(i)}$: total word count of class j

The resulting classification rule for a new data point $\mathbf{x} = (x_1, \ldots, x_d)'$ is

$$\begin{aligned}\hat{j} &= \text{argmax}_j \, \hat{\pi}_j f_j(\mathbf{x}) \\ &= \text{argmax}_j \, \log \hat{\pi}_j + \sum_{s=1}^{d} x_s \log \hat{p}_{sj} \\ &= \text{argmax}_j \, \mathbf{w}_j^T \cdot \mathbf{x} + b_j\end{aligned}$$

where

$$\mathbf{w}_j = \left(\log \hat{p}_{1j}, \ldots, \log \hat{p}_{dj}\right)^T, \quad b_j = \log \hat{\pi}_j.$$

This shows that like Bernoulli naive Bayes, multinomial naive Bayes is also a linear classifier.

Comments on multinomial naive Bayes[2]

Overall, naive Bayes is not naive!

- Very fast, low storage requirements
- Robust to irrelevant features
- Very good in domains with many equally important features
- Optimal Bayes classifier if the independence assumptions hold
- A good dependable baseline for text classification

[2] https://web.stanford.edu/~jurafsky/slp3/slides/7_NB.pdf

Multivariate multinomial naive Bayes

For categorical features $\vec{X} = (X_1, \ldots, X_d)$, they are assumed to be independent. We then fit a joint model for them within each class.

Suppose X_s has L distinct levels, which occur in class j with probabilities

$$P(X_s = \ell \mid Y = j) = p_{sj}^{(\ell)}, \quad \ell = 1, \ldots, L$$

It can be shown that the MLE of $p_{sj}^{(\ell)}$ is the observed fraction of level ℓ of X_s in class j. Then for a data point $\mathbf{x} = (x_1, \ldots, x_d)'$, the decision rule is

$$\hat{j} = \text{argmax}_j \, \hat{\pi}_j \prod_{s=1}^{d} P(X_s = x_s \mid Y = j) = \text{argmax}_j \, \hat{\pi}_j \prod_{s=1}^{d} \hat{p}_{sj}^{(x_s)}$$

Python scripts for naive Bayes

See

`https://scikit-learn.org/stable/modules/classes.html#module-sklearn.naive_bayes`