LEC 4: Bayes classifiers

Guangliang Chen

October 3, 2018

Outline

- Bayes classifiers: A family of classifiers based on posterior probabilities

$$\hat{y} = \arg\max_j P(\mathbf{x} \in C_j \mid \mathbf{x})$$

- kNN is a Bayes classifier (nonparametric)

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

- Other Bayes classifiers covered in this course
  - LDA / QDA
  - Naive Bayes

Introduction to Bayes classification

Today we look at a class of classifiers that are based on the Bayes' Rule:

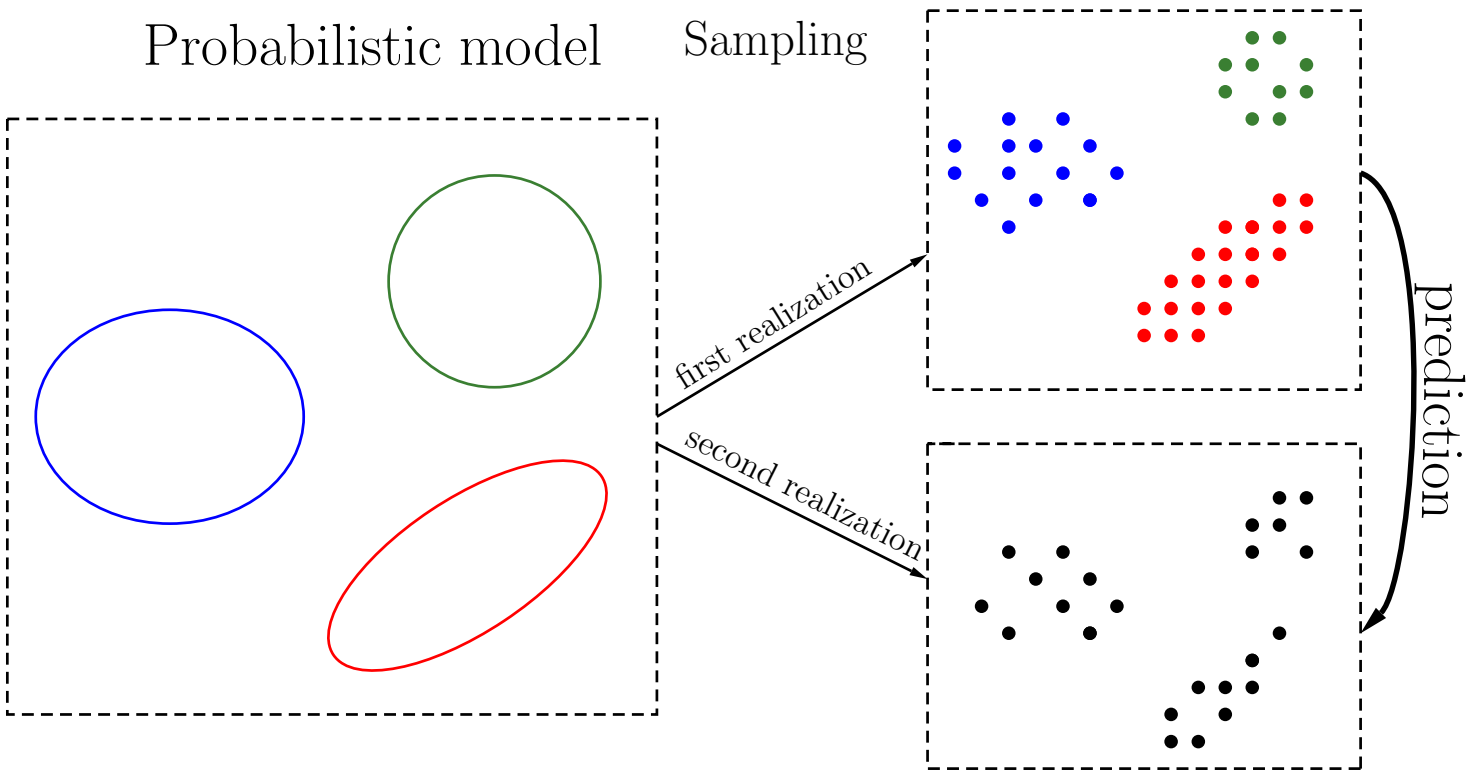
$$P(A_i \mid B) = \frac{P(B \mid A_i)P(A_i)}{\sum_{j=1}^{c} P(B \mid A_j)P(A_j)}, \quad \text{for each } i = 1, \ldots, c$$

where $A_1, \ldots, A_c$ are disjoint events that form a partition of the sample space.

In the above,

- $P(A_j), 1 \leq j \leq c$: prior probabilities
- $P(B \mid A_j), 1 \leq j \leq c$: conditional probabilities
- $P(A_i \mid B), 1 \leq j \leq c$: posterior probability of event A_i occurring (given that event B has occurred)

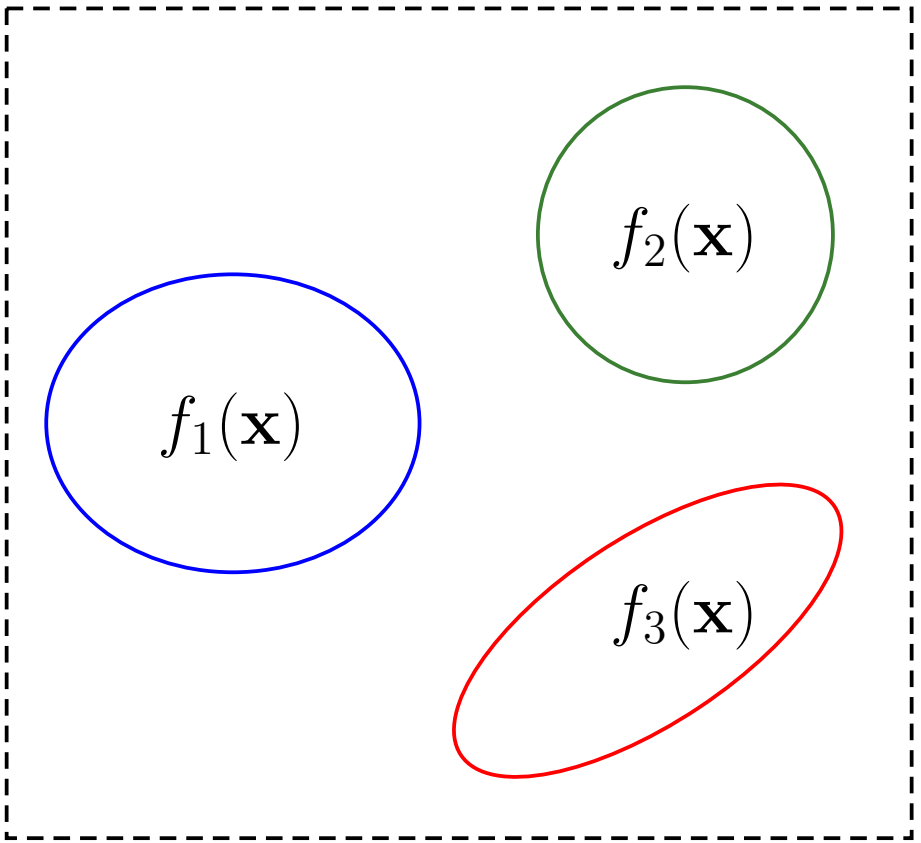
To apply Bayes' rule in the setting of classification, we first need to introduce a probabilistic model, i.e., a distribution, from which the training and test data are assumed to be obtained as two independent random samples.



Probabilistic models

We present a mixture model for the underlying distribution (of both training and test data):

- We model the distribution of each training class C_j by a pdf $f_j(\mathbf{x})$.
- We assume that the sampling frequency from each training class C_j is π_j ($\pi_j > 0, \sum \pi_j = 1$), i.e., for a fraction π_j of the time, $\mathbf{x}$ is sampled from C_j.



The *Law of Total Probability* implies that the mixture distribution has a combined density function

$$f(\mathbf{x}) = \sum f(\mathbf{x} \mid \mathbf{x} \in C_j) \cdot P(\mathbf{x} \in C_j) = \sum f_j(\mathbf{x}) \cdot \pi_j.$$

The training and test data represent two independent samples from the distribution $f(\mathbf{x})$.

We call $\pi_j = P(\mathbf{x} \in C_j)$ the *prior probability* of $\mathbf{x} \in C_j$, i.e., probability that a new sample $\mathbf{x}$ belongs to C_j before it is seen.

For example, if $\pi_1 = 0.3, \pi_2 = 0.5, \pi_3 = 0.2$, then the prior probability of a new sample belonging to class 1 is 0.3, and that of a new sample belonging to class 2 is 0.5, etc.

How to classify a new sample

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

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

We don't know the true values of π_j, so we'll estimate them using the observed training classes (in fact, only their sizes):

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

This method makes constant prediction (toward the largest training class in size), with associated error rate $1 - \frac{n_{\hat{j}}}{n}$. For example, for a training set of 3 classes with sizes 100, 100, and 150, the error rate would be $1 - \frac{150}{350} = \frac{4}{7}$.

What is a better way?

Bayes classification

A (much) better way is to assign the label based on the **posterior probabilities** (i.e., probabilities that a new data point belongs to the classes after we see it):

$$\hat{j} = \text{argmax}_j \; P(\mathbf{x} \in C_j \mid \mathbf{x})$$

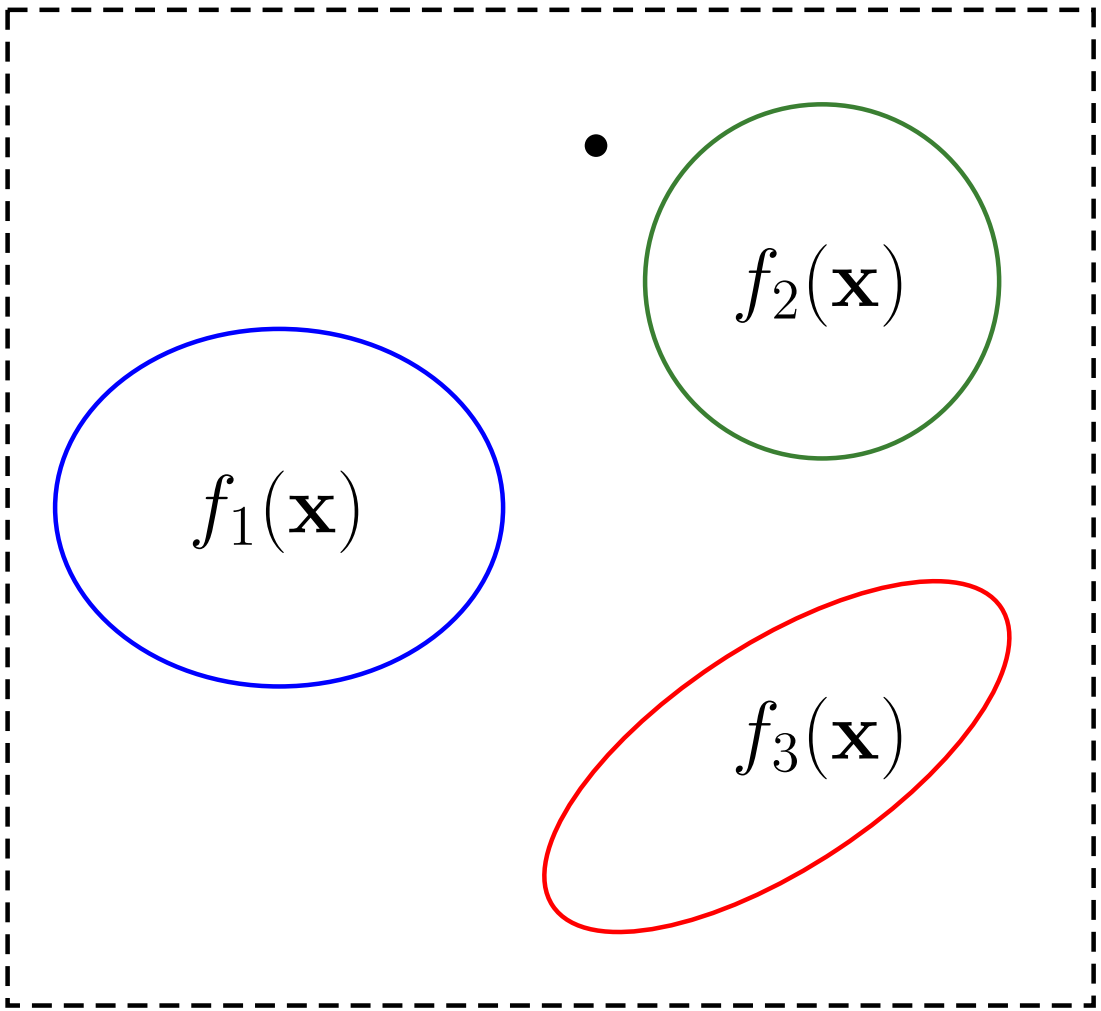
According to *Bayes' Rule*, the posterior probabilities are given by

$$P(\mathbf{x} \in C_j \mid \mathbf{x}) = \frac{f(\mathbf{x} \mid \mathbf{x} \in C_j) \cdot P(\mathbf{x} \in C_j)}{f(\mathbf{x})} \propto f_j(\mathbf{x})\pi_j$$

Therefore, the Bayes classification rule can be stated as

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

$\longleftarrow$ generic Bayes classifier


$f_2(\mathbf{x})$
$f_1(\mathbf{x})$
$f_3(\mathbf{x})$

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

To specify the component distributions $f_j(\mathbf{x})$, we pick a common distribution type (such as Gaussian) but combine it with different parameter values.

Different choices of the model lead to different Bayes classifiers:

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

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

- **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

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

in higher dimensions:

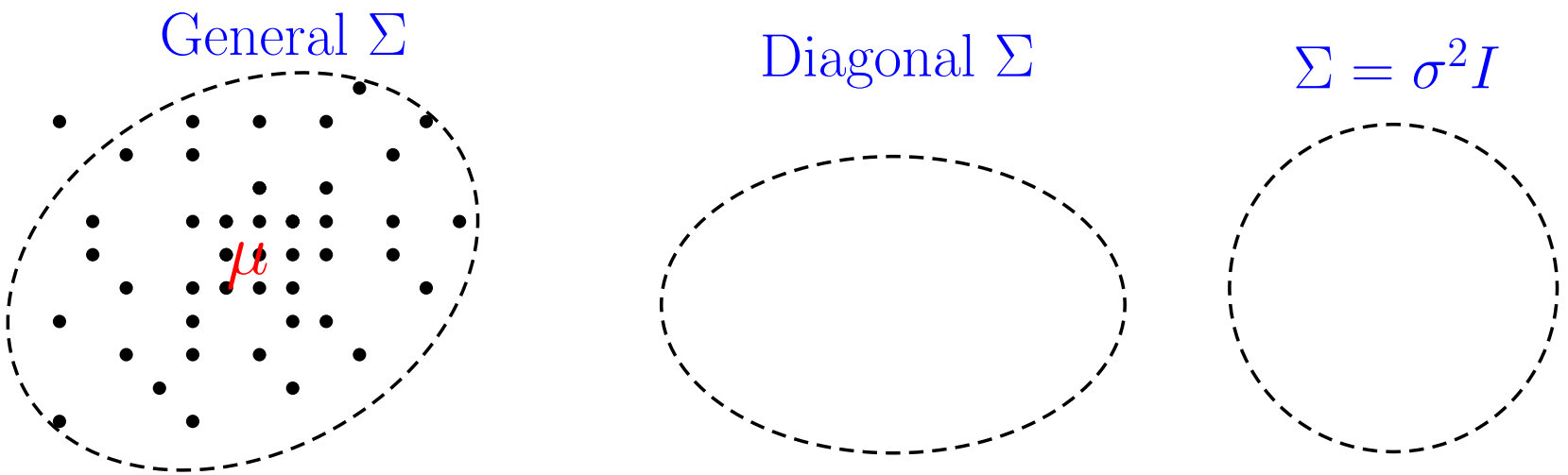
$$f(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}, \quad \forall\ \mathbf{x} \in \mathbb{R}^d$$

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

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

In the pdf of a multivariate Gaussian,

- $\boldsymbol{\mu} = \mathrm{E}(\mathbf{x}) \in \mathbb{R}^d$: center of the distribution
- $\boldsymbol{\Sigma} = \mathrm{E}((\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T) \in \mathbb{R}^{d \times d}$: covariance matrix



The Bivariate case ($d = 2$)

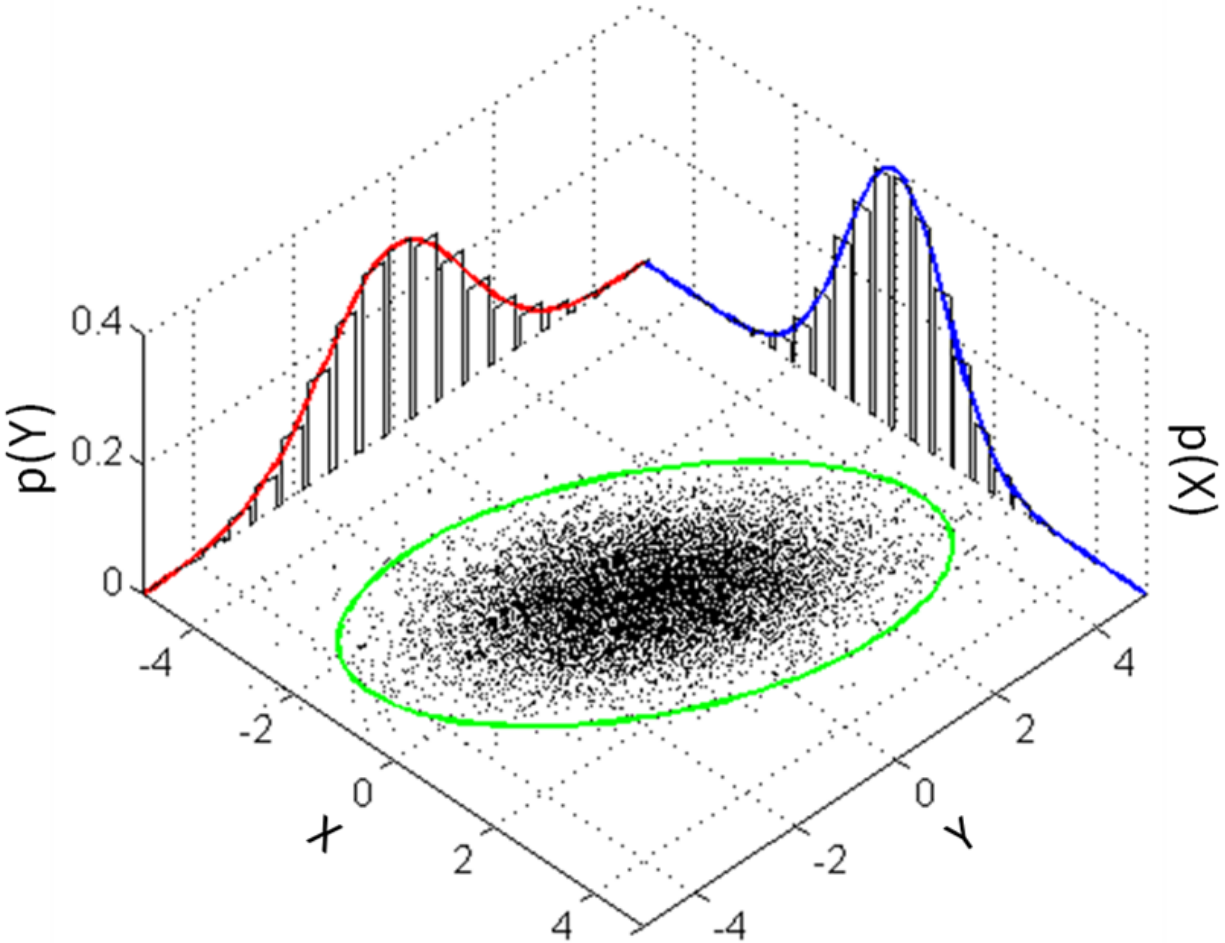
$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \boldsymbol{\mu} = \begin{pmatrix} m_1 \\ m_2 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$$

In this case, the joint density is

$$f(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \cdot \exp\left(-\frac{1}{2(1-\rho^2)}\left[\frac{(x_1-m_1)^2}{\sigma_1^2} + \frac{(x_2-m_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-m_1)(x_2-m_2)}{\sigma_1\sigma_2}\right]\right)$$

Here, m_i, σ_i^2 are mean and variance of x_i, and ρ is the correlation between x_1, x_2.

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



Bayes classification with multivariate Gaussians

Under such a mixture of Gaussians model,

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

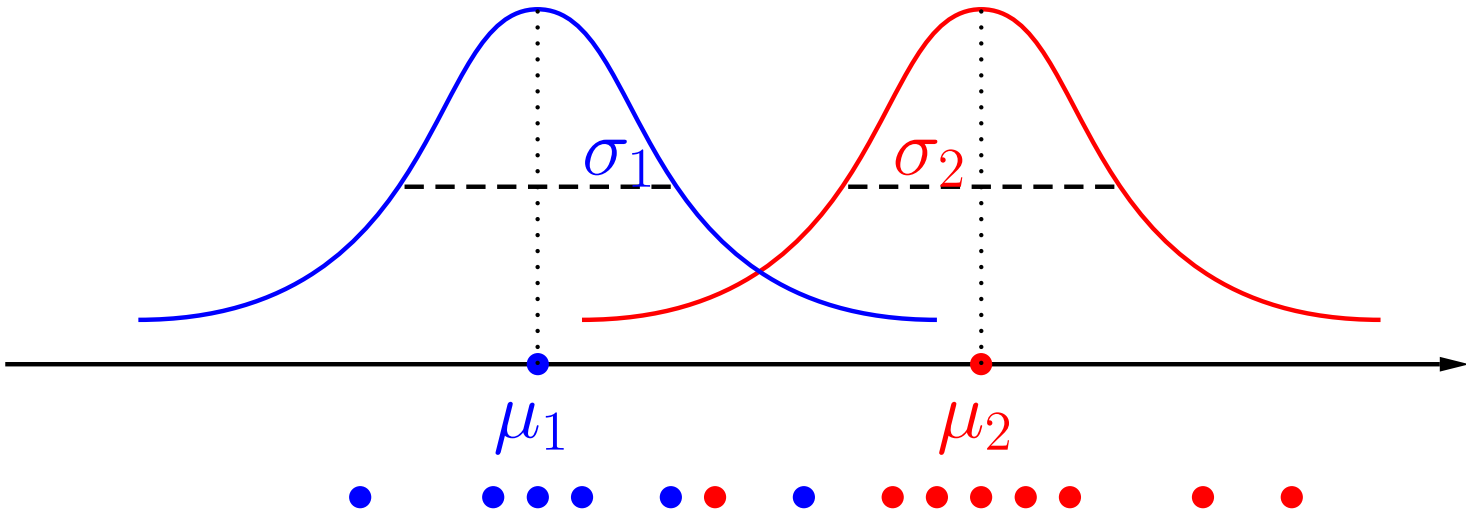
the Bayes classification rule (for a new point $\mathbf{x}$)

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

becomes the following:

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

Example 0.1. 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

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

Two remarks:

- 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 the Bayes classifier has quadratic boundaries (between each pair of training classes).

We call the above classifier *Quadratic Discriminant Analysis (QDA)*.

Parameter estimation for QDA

The formulation of the QDA classifier

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

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

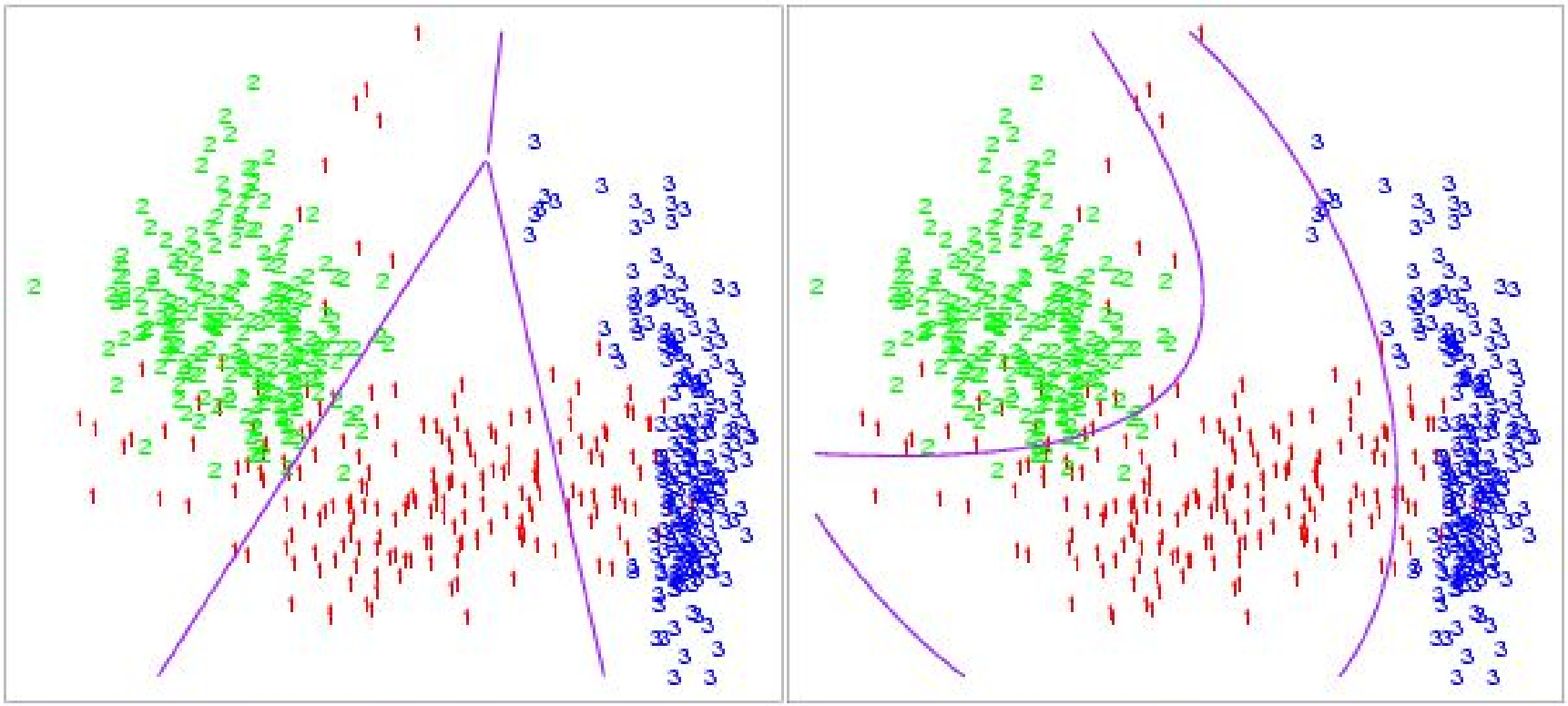
Given training data, we estimate them as follows:

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

Thus, the practical QDA classification rule is

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

LDA (left) and QDA (right)



The case of equal covariance

QDA assumes that the component distributions are all multivariate Gaussian (but with separate means $\boldsymbol{\mu}_j$ and covariances $\boldsymbol{\Sigma}_j$).

However, there are a lot of parameters that need to be estimated from the training data (especially when in very high dimensions)! $\longleftarrow$ There is also a risk of overfitting

To ease the computational burden, we assume that

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

so that the different component distributions are just shifted versions of each other (i.e., same covariance, different centers).

In this case, 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 of the equal-covariance classifier is:

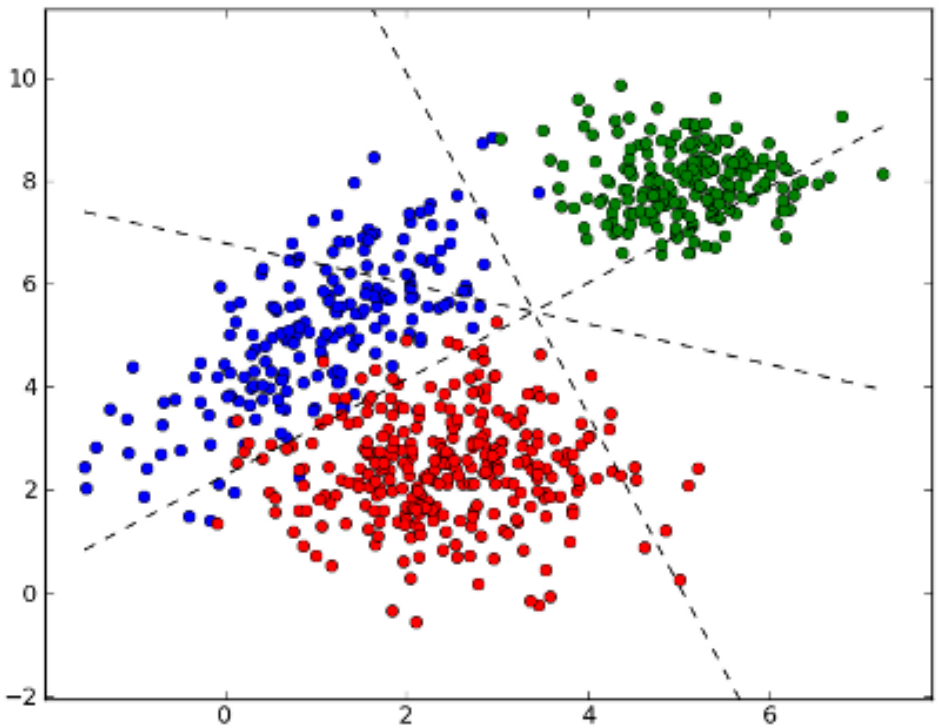
$$\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_j - \frac{1}{2}\boldsymbol{\mu}_j^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_j + \log \pi_j = \mathbf{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_\ell - \frac{1}{2}\boldsymbol{\mu}_\ell^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_\ell + \log \pi_\ell$$

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 $\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_\ell)$, showing that the classifier has linear boundaries.

We call it the *Linear Discriminant Analysis (LDA)* classifier.



Source: `http://mlpy.sourceforge.net/docs/3.5/lin_class.html`

Parameter estimation for LDA

Similarly, we can use the training data to estimate the LDA parameters $\pi_j, \boldsymbol{\mu}_j$ in the same way as before:

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

However, for the common covariance matrix $\boldsymbol{\Sigma}$, we use the following pooled estimator:

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

This leads to the following practical LDA classifier:

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

When statistics meets optimization

We have introduced LDA both as a supervised dimensionality reduction approach and as a Bayes classifier. This is not a conflict.

We show this in the two-class setting, where the decision boundary of the LDA classifier is a hyperplane with normal vector $\hat{\boldsymbol{\Sigma}}^{-1}(\hat{\boldsymbol{\mu}}_1 - \hat{\boldsymbol{\mu}}_2)$.

As a dimensionality reduction technique, LDA projects the data onto the following direction

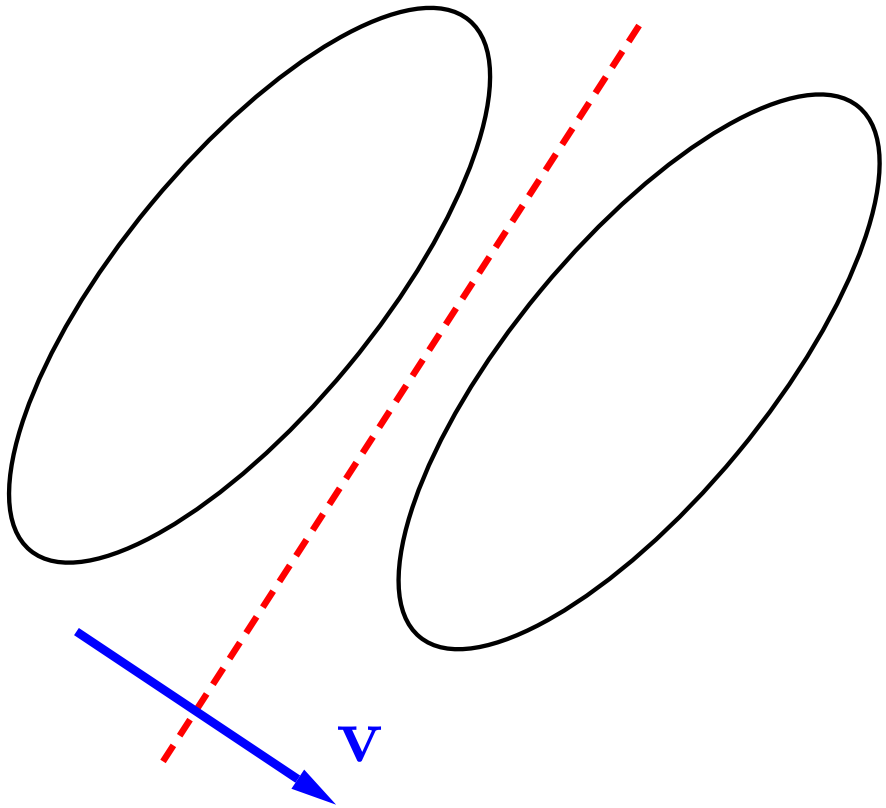
$$\mathbf{v} = \mathbf{S}_w^{-1}(\mathbf{m}_1 - \mathbf{m}_2) \longleftarrow \text{Same as the above normal vector}$$

where

$$\mathbf{S}_w = \sum_{j=1}^{2} \sum_{\mathbf{x} \in C_j} (\mathbf{x} - \mathbf{m}_j)(\mathbf{x} - \mathbf{m}_j)^T = (n-2)\hat{\boldsymbol{\Sigma}}, \quad \mathbf{m}_1 = \hat{\boldsymbol{\mu}}_1, \ \mathbf{m}_2 = \hat{\boldsymbol{\mu}}_2.$$

Therefore, we can combine both perspectives to fully understand LDA:

- A Linear classifier
- Assuming a mixture of Gaussians model (with equal covariance) when used as a classifier
- Based on Bayes' rule
- Separating training data along the optimal discriminatory direction
- As a projection method, it is applicable to more general data.



MATLAB implementation of LDA/QDA

% fit a discriminant analysis classifier

mdl = fitcdiscr(trainData, trainLabels, 'DiscrimType', type)

% where **type** is one of the following:

- **'Linear'** (default): LDA
- **'Quadratic'**: QDA

% classify new data

pred = predict(mdl, testData)

Python scripts for LDA/QDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

#from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()

pred = lda.fit(trainData,trainLabels).predict(testData)

print("Number of mislabeled points: %d" %(testLabels != pred).sum())
```

The singularity issue in LDA/QDA

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

Common techniques to fix this:

- Apply PCA to reduce dimensionality first, or
- Regularize the covariance matrices, or
- Use psuedoinverse: 'pseudoLinear', 'pseudoQuadratic':

$$(\mathbf{S}_w)^\dagger = (\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T)^\dagger = \mathbf{Q}\mathbf{\Lambda}^\dagger\mathbf{Q}^T, \quad \mathbf{\Lambda}^\dagger_{ii} = \begin{cases} \frac{1}{\lambda}{}_i, & \lambda_i > 0 \\ 0, & \lambda_i = 0 \end{cases}$$

Naive Bayes

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

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

However, a simplifying assumption is made on the individual features of $\mathbf{x}$:

$$f_j(\mathbf{x}) = \prod_{k=1}^{d} f_{jk}(x_k) \qquad (x_1, \ldots, x_d \text{ are independent})$$

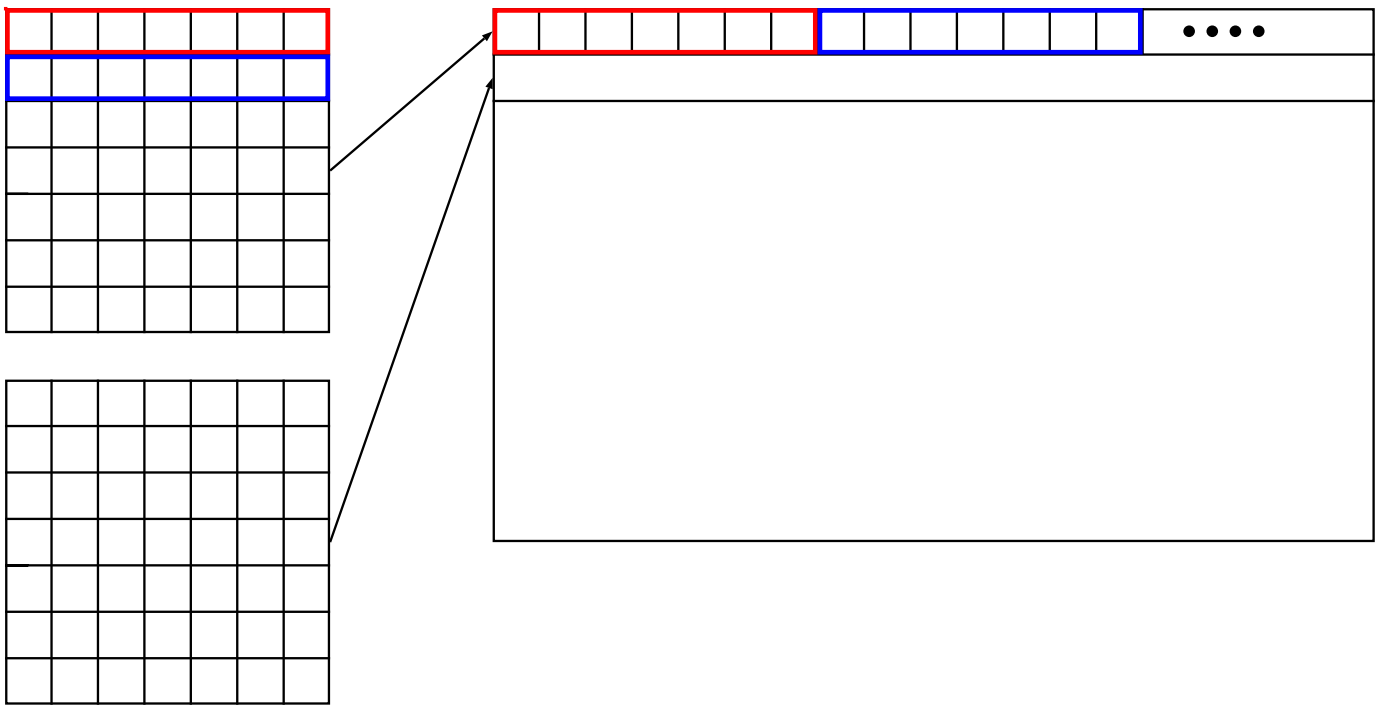
such that the classification rule becomes

$$\hat{j} = \text{argmax}_j \ \pi_j \prod_{k=1}^{d} f_{jk}(x_k) = \text{argmax}_j \ \log \pi_j + \sum_{k=1}^{d} \log f_{jk}(x_k)$$

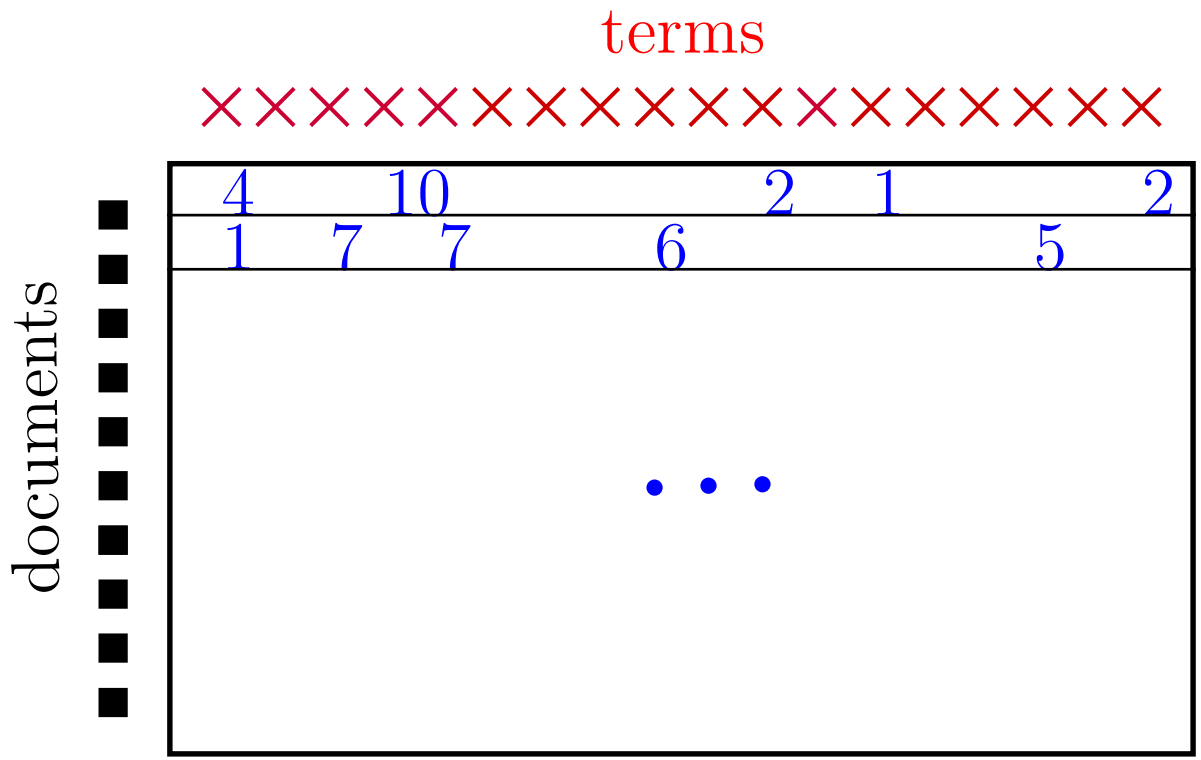
where f_{jk} remain to be specified.

Two applications

- Classification of digital images: In this case, each pixel is a feature, so naive Bayes assumes that pixel intensities are independent random variables.



- Classification of text documents: Each term defines a feature and their frequencies are assumed to be independent random variables.



(Bag of words model for text corpus)

How to estimate f_{ij}

The independence assumption reduces the high dimensional density estimation problem ($\{f_j(\mathbf{x})\}_j$) to a union of simple 1D problems ($\{f_{jk}(x_k)\}_{j,k}$).

Again, we need to pick a model for the 1D distributions f_{jk}.

For continuous features (e.g., image data), the standard choice is the 1D normal distribution

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

where μ_{jk}, σ_{jk} can be estimated similarly using the training data. With such a choice, the classifier (which is called Gaussian naive Bayes) is

$$\hat{j} = \text{argmax}_j \ \log \pi_j - \sum_{k=1}^{d} \left[\log \sigma_{jk} + (x_k - \mu_{jk})^2/2\sigma_{jk}^2\right]$$

Other cases:

- For categorical features (e.g., sex, education level, etc.), we can use the binomial/multinomial distribution to model such a feature
- For count-based data (such as the bag-of-words model for text documents), we can also use the multinomial distribution to model the frequency counts of different terms in a document

Such classifiers are called multinomial naive Bayes (and they can become the topic of one or two final projects).

MATLAB functions for Naive Bayes

% fit a naive Bayes classifier

mdl = fitcnb(trainData, trainLabels, 'Distribution', 'normal')

% classify new data

pred = predict(mdl, testData)

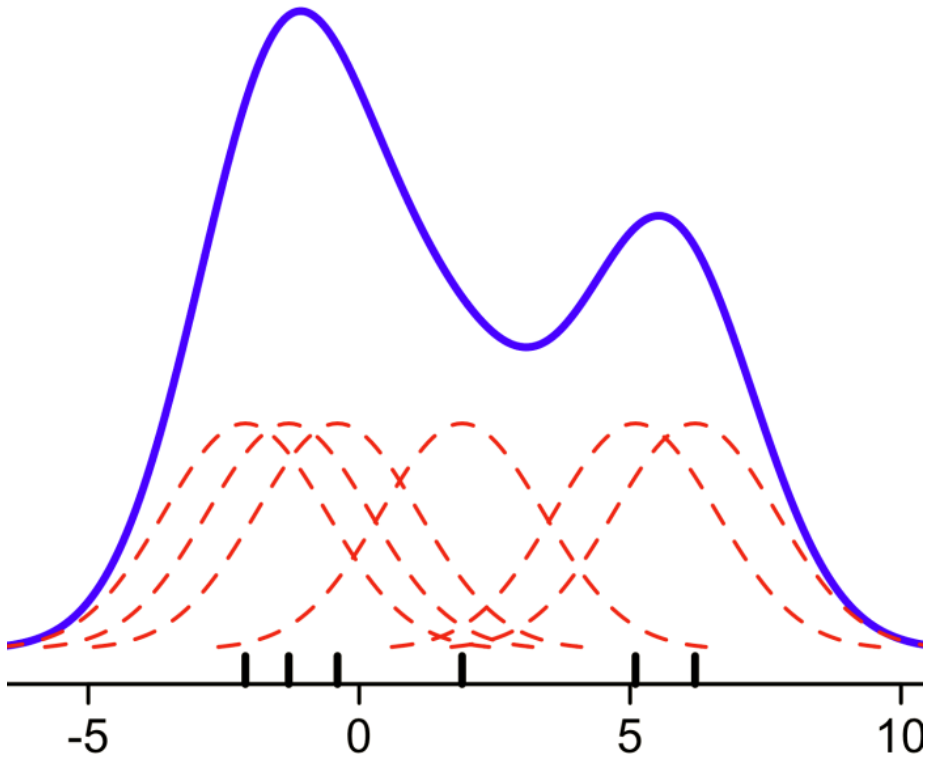
Python scripts for Naive Bayes (for continuous features)

```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

pred = gnb.fit(trainData, trainLabels).predict(testData)

print("Number of mislabeled points: %d" %(testLabels != pred).sum())
```

Improving Naive Bayes

Independence assumption: apply PCA to get uncorrelated features (closer to being independent)

Choice of distribution: can use kernel smoothing to be more flexible
mdl = fitcnb(trainData, trainLabels, 'Distribution', 'kernel')
However, this will be at the expense of speed.



Bayes classification: A summary

- General decision rule

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

- Examples of Bayes classifiers
  - **QDA**: multivariate Gaussians
  - **LDA**: multivariate Gaussians with equal covariance
  - **Naive Bayes**: independent features $x_1, \ldots, x_d$

HW4 (due October 27, Saturday noon)

This is a group assignment, meaning that you can work on the problems either alone or with a partner (in the latter case, one submission is enough, but obviously you need to indicate who you collaborated with).

1. First use PCA to project the USPS data set (both training and test) into s dimensions (for each choice below) and then use the LDA classifer to classify the test data. Present and discuss your results (you may want to compare with kNN and NLC when also combined with PCA).
   - $s = 88$ (95% variance)
   - $s =$ your own choice (preferably better than 88)
2. Repeat Question 1 with QDA instead of LDA. Is it better than LDA?

3. In the setting of Question 1 apply now the Naive Bayes classifier (by fitting pixelwise normal distributions) with the following choices of s:

   - $s = 256$ (no projection)
   - $s = 88$ (95% variance)
   - $s =$ your own choice

   What are your results like (when comparing with LDA and QDA)?

4. Apply PCA + LDA/QDA to the following subsets of the MNIST data set:

   - (a) 0, 1 (b) 4, 9 (c) 0, 1, 2 and (d) 3, 5, 8

   Report your results.

5. I have simulated a data set consisting of two bivariate Gaussians with nearly the same covariance (In the plot below, red and green colors code the two training classes and black points are the test data):

   It has been listed on the course webpage, so you can go there and download it for homework use.

   Apply LDA and QDA separately to classify the test data and report their error rates. Can you add the decision boundary of each classifier to the plot?