Bayes classifiers
Outline

• A probabilistic model for classification
• Bayes classification rule
• Examples of Bayes classifiers
  – Discriminant Analysis: LDA, QDA
  – Naive Bayes: Gaussian, Bernoulli, Multinomial, Multivariate multinomial
• Assignment 3
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 $Y$ for each of the test point locations $\vec{X}_{n+j}$, $1 \leq j \leq m$. 
How to classify a new sample naively

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 $\pi_j$ 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

$$\hat{j} = \arg\max_j \pi_j$$

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

We don’t know the true values of $\pi_j$, so we’ll estimate them using a random sample from the joint distribution:

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

Let the count of the observations in the above sample that come from class $j$ be

$$N_j = \sum_{i=1}^{n} I(Y_i = j), \quad 1 \leq j \leq c$$

Then

$$E(N_j) = \sum_{i=1}^{n} E[I(Y_i = j)] = \sum_{i=1}^{n} (1 \cdot \pi_j + 0 \cdot (1 - \pi_j)) = n \pi_j$$

This shows that $N_j/n$ is an unbiased estimator of $\pi_j$. 
Bayes classifiers

When given a specific training data set:

\[(x_1, y_1), \ldots, (x_n, y_n),\]

a point estimate of \(\pi_j\) is thus

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

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

\[n_j = \sum_{i=1}^{n} I(y_i = j), \quad 1 \leq j \leq c\]
Bayes classification rule

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

\[
\hat{j} = \arg \max_j P(Y = j \mid \vec{X} = x) \leftarrow \text{generic Bayes classifier}
\]

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

\[
1 - E_{\vec{X}} \left( \max_{1 \leq j \leq c} P(Y = j \mid \vec{X}) \right)
\]

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):

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

Model-based Bayes classification

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

$$f(x \mid Y = j) = f_j(x).$$

The joint distribution of $\vec{X}$ and $Y$ is

$$f_{\vec{X}, Y}(x, Y = j) = f(x \mid Y = j) \ P(Y = j)$$

$$= f_j(x) \ \pi_j$$

and the marginal density of $\vec{X}$ is

$$f_{\vec{X}}(x) = \sum_j f_{\vec{X}, Y}(x, Y = j) = \sum_j \pi_j f_j(x)$$
According to Bayes’ rule, the posterior probabilities are given by

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

Therefore, the Bayes classification rule can be stated as

$$\hat{j} = \arg\max_j f_j(x) \cdot \pi_j \quad \text{← model-based Bayes classifier}$$
Bayes classifiers

Estimating class-conditional probabilities $f_j(x)$

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

Different kinds of $f_j(x)$ lead to different Bayes classifiers:

- **LDA/QDA** - multivariate Gaussian distributions
  
  $f_j(x) = \frac{1}{(2\pi)^{d/2}|\Sigma_j|^{1/2}} e^{-\frac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j)}, \quad j = 1, \ldots, c$

- **Naive Bayes** - by assuming independent features in $x = (x_1, \ldots, x_d)$:
  
  $f_j(x) = \prod_{k=1}^{d} f_{jk}(x_k) \leftarrow 1D$ 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:

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

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

\[ f(x) = \frac{1}{(2\pi \sigma^2)^{d/2}} e^{-\frac{\|x-\mu\|^2}{2\sigma^2}}, \quad \forall \ x \in \mathbb{R}^d \]
In the pdf of a multivariate Gaussian,

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

General \( \Sigma \) \n
Diagonal \( \Sigma \) \n
\[ \Sigma = \sigma^2 I \]
**The Bivariate normal \((d = 2)\)**

Write

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

The joint density is

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

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,

\[ f_j(x) = \frac{1}{(2\pi)^{d/2}|\Sigma_j|^{1/2}} e^{-\frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1}(x-\mu_j)}, \quad \forall \ j = 1, \ldots, c \]

the Bayes classification rule (for a new data point \( x \))

\[ \hat{j} = \arg\max_j f_j(x) \pi_j \]

becomes the following:

\[ \hat{j} = \arg\max_j \frac{1}{(2\pi)^{d/2}|\Sigma_j|^{1/2}} e^{-\frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1}(x-\mu_j)} \cdot \pi_j \]

\[ = \arg\max_j \log \pi_j - \frac{1}{2} \log |\Sigma_j| - \frac{1}{2}(x - \mu_j)^T \Sigma_j^{-1}(x - \mu_j) \]
Bayes classifiers

Example 0.1. Let’s consider the special case of two 1D Gaussians:

\[
\hat{j} = \arg\max_j \log \pi_j - \frac{1}{2} \log(\sigma_j^2) - \frac{(x - \mu_j)^2}{2\sigma_j^2}
\]

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

- If \( \pi_1 = \pi_2 \) and \( \sigma_1 = \sigma_2 \), then the rule will assign \( x \) to the closer mean \( \mu_j \) (larger \( \pi_j \) 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

\[
\log \pi_j - \frac{1}{2} \log |\Sigma_j| - \frac{1}{2} (x - \mu_j)^T \Sigma_j^{-1} (x - \mu_j)
\]

\[
= \log \pi_\ell - \frac{1}{2} \log |\Sigma_\ell| - \frac{1}{2} (x - \mu_\ell)^T \Sigma_\ell^{-1} (x - \mu_\ell)
\]

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

Parameter estimation for QDA

The QDA classifier

\[ \hat{j} = \arg \max_j \log \pi_j - \frac{1}{2} \log |\Sigma_j| - \frac{1}{2} (x - \mu_j)^T \Sigma_j^{-1} (x - \mu_j) \]

depends on the model parameters \( \pi_j, \mu_j, \Sigma_j \) but their true values are typically unknown.

Given training data, we estimate them as follows:

\[ \hat{\pi}_j = \frac{n_j}{n}, \quad \hat{\mu}_j = \frac{1}{n_j} \sum_{x \in C_j} x, \quad \hat{\Sigma}_j = \frac{1}{n_j - 1} \sum_{x \in C_j} (x - \hat{\mu}_j)(x - \hat{\mu}_j)^T \]
Bayes classifiers

LDA (left) and QDA (right)
Linear Discriminant Analysis (LDA)

In QDA we assume that the component distributions are all multivariate Gaussians but allow them to have separate means $\mu_j$ and covariances $\Sigma_j$.

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

$$\Sigma_1 = \cdots = \Sigma_c = \Sigma$$
In this special setting, the Bayes classification rule becomes

$$\hat{j} = \arg\max_j \log \pi_j - \frac{1}{2} (\mathbf{x} - \mathbf{\mu}_j)^T \Sigma^{-1} (\mathbf{x} - \mathbf{\mu}_j)$$

$$= \arg\max_j \mathbf{x}^T \Sigma^{-1} \mathbf{\mu}_j - \frac{1}{2} \mathbf{\mu}_j^T \Sigma^{-1} \mathbf{\mu}_j + \log \pi_j.$$ 

The corresponding decision boundary is

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

which simplifies to

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

This is a hyperplane with normal vector $\Sigma^{-1}(\mathbf{\mu}_j - \mathbf{\mu}_\ell)$. 
Parameter estimation for LDA

Similarly, we can use the training data to estimate the parameters $\pi_j, \mu_j$ as follows:

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

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

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

This leads to the following practical LDA classifier:

$$\hat{j} = \arg\max_j x^T \hat{\Sigma}^{-1} \hat{\mu}_j - \frac{1}{2} \hat{\mu}_j^T \hat{\Sigma}^{-1} \hat{\mu}_j + \log \hat{\pi}_j.$$
When statistics meets optimization

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

In the two-class setting,

- As a classifier, the LDA decision boundary is a hyperplane with normal vector \( \hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_2) \).

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

\[
v = S_w^{-1}(m_1 - m_2)
\]
Bayes classifiers

It turns out that the two vectors are the same:

\[
S_w = \sum_{j=1}^{2} \sum_{x \in C_j} (x - m_j)(x - m_j)^T = (n - 2)\hat{\Sigma}
\]

\[
m_1 = \hat{\mu}_1
\]

\[
m_2 = \hat{\mu}_2
\]

This shows that the two LDAs are indeed the same thing.

Therefore, we can combine both perspectives to fully understand LDA (see next slide).
**Remark.** LDA

- is a linear classifier (which classifies data using a hyperplane)
- uses a mixture of Gaussians model (with equal covariance)
- is a Bayes classifier
- projects data onto the most discriminative direction
- is also applicable to data from other distributions
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)
Bayes classifiers

Python scripts for LDA/QDA

```python
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 (nearly) singular in the case of high dimensional data.

Common fixes:

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

\[
(S_w)^\dagger = (Q \Lambda Q^T)^\dagger = Q \Lambda^\dagger Q^T, \quad \Lambda^\dagger_{ii} = \begin{cases} 
\frac{1}{\lambda_i}, & \lambda_i > 0 \\
0, & \lambda_i = 0 
\end{cases}
\]
LDA/QDA on the MNIST

See poster at

https://www.sjsu.edu/faculty/guangliang.chen/Math285S16/poster-DA.pdf
Naive Bayes

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

$$\hat{j} = \arg\max_j f_j(x)\pi_j$$

Unlike Discriminant Analysis (LDA/QDA) which assumes that $\vec{X}$, when conditioned on $Y = j$, follows a multivariate Gaussian distribution $f_j(x)$, Naive Bayes assumes that the individual components of $\vec{X} = (X_1, \ldots X_d)$ are conditionally independent given a class $Y = j$:

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

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

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
Bayes classifiers

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:

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

where \( \mu_{sj}, \sigma_{sj} \) 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 \( x = (x_1, \ldots, x_d)' \) is

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

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

$$\hat{j} = \text{argmax}_j \log \hat{\pi}_j + \sum_{s=1}^{d} [x_s \log \hat{p}_{sj} + (1 - x_s) \log(1 - \hat{p}_{sj})]$$
Remark. Bernoulli naive Bayes is a linear classifier, because for each \( j \),

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

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

\[
= \mathbf{w}_j^T \cdot \mathbf{x} + b_j
\]

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).
Bayes classifiers

Multinomial naive Bayes

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

```
terms

×××××××××××××××××××××××××

4  10  5  1  2
1  7  7  6

...
Bayes classifiers

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:

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

where $j$ is fixed and

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

are the probabilities of generating terms from a fixed topic.
Bayes classifiers

It can be shown that the MLE of \( p_{sj} \) based on the \( j \)th training class is

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

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

- \( \sum_{i=1}^{n_j} x_s^{(i)} \): word count of \( s \)th 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

\[
\hat{j} = \arg\max_j \hat{\pi}_j f_j(\mathbf{x})
\]

\[
= \arg\max_j \log \hat{\pi}_j + \sum_{s=1}^{d} x_s \log \hat{p}_{sj}
\]

\[
= \arg\max_j \mathbf{w}_j^T \cdot \mathbf{x} + b_j
\]

where

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

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

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

---

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 $\ell$ of $X_s$ in class $j$. Then for a data point $x = (x_1, \ldots, x_d)'$, the decision rule is

$$\hat{j} = \arg\max_j \hat{\pi}_j \prod_{s=1}^{d} P(X_s = x_s \mid Y = j) = \arg\max_j \hat{\pi}_j \prod_{s=1}^{d} \hat{p}_{sj}^{(x_s)}$$
MATLAB function for naive Bayes

% fit a Gaussian naive Bayes classifier
mdl = fitcnb(trainData, trainLabels, 'DistributionNames', 'normal')
% fit a naive Bayes classifier with kernel smoothing
mdl = fitcnb(trainData, trainLabels, 'DistributionNames', 'kernel')
% fit a multinomial naive Bayes classifier
mdl = fitcnb(trainData, trainLabels, 'DistributionNames', 'mn')
% fit a multivariate multinomial naive Bayes classifier
mdl = fitcnb(trainData, trainLabels, 'DistributionNames', 'mvmn')

% classify new data
pred = predict(mdl, testData)
Python scripts for naive Bayes

See

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

Summary

- Bayes decision rule

$$\hat{j} = \arg\max_j P(Y = j | \mathbf{x})$$

- Examples of Bayes classifiers
  - **QDA**: multivariate Gaussians
  - **LDA**: multivariate Gaussians with equal covariance
  - **Naive Bayes**: independent features $x_1, \ldots, x_d$
Assignment 3

1. Use PCA with each value of $s = s_{\text{min}} (80\%) : s_{\text{max}} (95\%)$ to project the Fashion-MNIST data set (both training and test) into an $s$-dimensional space and then apply each of the following classifiers to classify the test data. Plot the test accurate rates for all the four methods as functions of $s$ in one figure and discuss your results.

- LDA
- QDA
- Gaussian Naive Bayes
- Naive Bayes with kernel smoothing