# Applied Statistical Methods

## Classification - Part II

Xuemao Zhang
East Stroudsburg University

April 12, 2023

# Outline

- Introduction
- Linear Discriminant Analysis (LDA)
  - LDA when $p = 1$
  - LDA when $p > 1$
- Quadratic Discriminant Analysis (QDA)
- Naive Bayes
- DA vs Logistic Regression
- Feature selection with CV

# Introduction

- Recall that in classification problem, we try to calculate

$$p_k(x) = P(Y = k|X = x), k = 1, \ldots, K.$$

  where $K$ is the number of elements in $\mathcal{C}$, the set of collection of responses of $Y$.

- Suppose we now have information on $f_k(x) = Pr(X = x|Y = k)$, feature distribution within each class,
- How do we use this to make predictions?

# Introduction

- We apply the Bayes Rule in probability:

**Bayes' Rule**

Let $S_1, S_2, \cdots, S_K$ be a partition of the sample space $S$ with **prior probabilities** $P(S_1), P(S_2), \cdots, P(S_K)$. Suppose an event $A$ occurs and $P(A|S_i)$ is known for each $i = 1, \ldots, K$. Then the **posterior probability** of $S_i$, given that $A$ occurred is

$$P(S_i|A) = \frac{P(A \cap S_i)}{P(A)} = \frac{P(S_i)P(A|S_i)}{\sum_{j=1}^{K} P(S_j)P(A|S_j)}, i = 1, \ldots, K.$$

# Introduction

- Bayes Theorem in our context is:

$$p_k(x) = Pr(Y = k|X = x) = \frac{Pr(X = x|Y = k) \cdot Pr(Y = k)}{Pr(X = x)}$$

or

$$p_k(x) = Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{i=1}^{K} \pi_i f_i(x)},$$

where in the formula, we need

- $f_k(x) = Pr(X = x|Y = k)$ which is the density for $X$ in class $k$, $k = 1, \ldots, K$
- $\pi_k = Pr(Y = k), k = 1, \ldots, K$ which is the marginal or **prior probability** of $Y$ for class $k$.
- We refer to $p_k(x)$ as the **posterior probability** that an observation posterior $X = x$ belongs to the *kth* class.

# Linear Discriminant Analysis

$\pi_k$ is generally simple to estimate:

- If our data are a random sample of size $n$, then we can use the sample proportion

$$\hat{\pi}_k = \frac{\#\{Y = k\}}{n},$$

  which is the fraction of the training observations that belong to the $k$th class.

- Otherwise can use outside information (eg. historical data)

# Linear Discriminant Analysis

$f_k(x)$ is much harder to estimate:

- Technically the notation $f_k(x) = Pr(X = x|Y = k)$ is correct only if $X$ is a discrete random variable. If $X$ is continuous, $f_k(x)dx$ would correspond to the probability of $X$ falling in in a small region $dx$ around $x$.
- Estimate of $f_k(x) = Pr(X = x|Y = k)$ is more difficult. This is a **density estimation** problem.
- In LDA (Linear Discriminant Analysis), we will use Gaussian/normal densities for these, separately in each class.

# Linear Discriminant Analysis when $p = 1$

- There is only one feature $X$.
- The Gaussian density has the form

$$\boxed{f_k(x) = \frac{1}{\sigma_k\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma_k}\right)^2}, -\infty < x < \infty.}$$

  Here $\mu_k$ is the mean, and $\sigma_k^2$ is the variance in class $k, k = 1, \ldots, K$.
- We will assume that all the $\sigma_k = \sigma$ are the same.
- Plugging this into Bayes formula,

$$p_k(x) = \frac{\pi_k \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma}\right)^2}}{\sum_{i=1}^{K} \pi_i \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_i}{\sigma}\right)^2}}, k = 1, \ldots, K.$$

  Happily, there are simplifications and cancellations.

# Linear Discriminant Analysis when $p = 1$

- To classify at the value $X = x$, we need to see which of the $p_k(x)$ is largest.
- Taking logs, and discarding terms that do not depend on $k$, we see that this is equivalent to assigning $x$ to the class with the largest **discriminant score**:

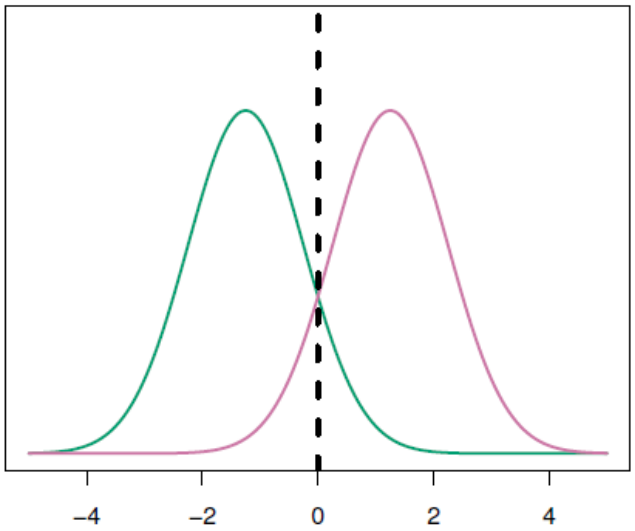
$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$$

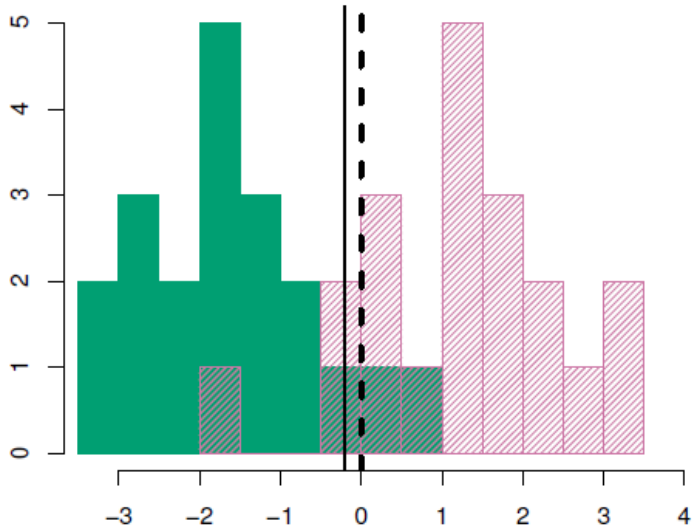
  which is a linear function of $x$.
- If there are $K = 2$ classes and $\pi_1 = \pi_2 = 0.5$, then one can show that the decision boundary is at

$$x = \frac{\mu_1 + \mu_2}{2}.$$

# Linear Discriminant Analysis when $p = 1$

- Example with $\mu_1 = -1.5, \mu_2 = 1.5$, $\pi_1 = \pi_2 = 0.5$, and $\sigma = 1$.




- Typically we don't know these parameters; we just have the training data. In that case we simply estimate the parameters and plug them into the rule.

# Linear Discriminant Analysis when $p = 1$

$$\hat{\pi}_k = \frac{n_k}{n}, \; n_k \text{ is the number of observations in class } k$$

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

$$\hat{\sigma}^2 = \frac{1}{n-k} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$$

$$= \frac{1}{n_k} \sum_{k=1}^{K} (n_k - 1) \hat{\sigma}_k^2,$$

where $\hat{\sigma}_k^2 = \frac{1}{n_k} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$ is the sample variance for the $k$th class. That is, $\hat{\sigma}^2$ is the pooled estimate of the common variance $\sigma^2$.

# Linear Discriminant Analysis when $p > 1$

- When $p > 1$, we consider multivariate normal distribution for $f_k(x) = Pr(X = x|Y = k), k = 1, \ldots, K$.
- To use matrix notation, we define the following matrices:

$$\mathbf{x} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{bmatrix}, \qquad \boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{bmatrix}$$

# Linear Discriminant Analysis when $p > 1$

**Definition.** A random vector

$$\mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{bmatrix}$$

is said to have a $MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ distribution if its pdf is given by

$$f(\mathbf{x}) = f(x_1, \ldots, x_p) = \left(\frac{1}{2\pi}\right)^{p/2} \left[\frac{1}{\det \boldsymbol{\Sigma}}\right]^{1/2} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right].$$

If $n = 2$, the distribution is called Bivariate Normal Distribution. Let $X_1$ and $X_2$ have a bivariate normal distribution, then

$$\boldsymbol{\mu} = (\mu_1, \mu_2)', \qquad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$$

# Linear Discriminant Analysis when $p > 1$

- linear correlation coefficient
  - The linear correlation coefficient of $X_1$ and $X_2$ is defined to be,

$$\boxed{\rho = \frac{Cov(X_1, X_2)}{\sigma_1 \sigma_2}}$$

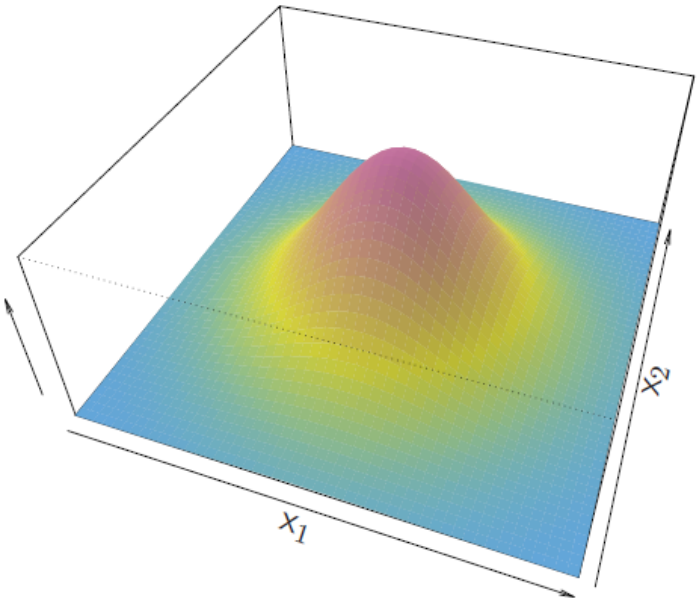
    where $\sigma_1$ and $\sigma_2$ are the standard deviations of $X_1$ and $X_2$, respectively.
  - $\boxed{-1 \leq \rho \leq 1}$
- Marginal distributions
  - Let $X \sim MVN(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The marginal distribution of any set of component $X$ is multivariate normal with means, variance and covariance obtained by taking the corresponding components of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ respectively.
  - Let $X_1$ and $X_2$ have a bivariate normal distribution. Then

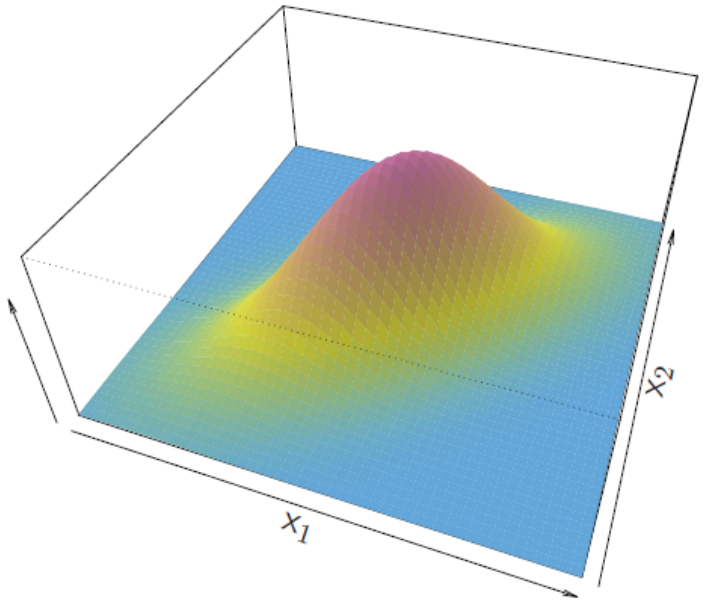
    **(a).** The marginal distribution of $X_1$ is normal with mean $\mu_1$ and variance $\sigma_1^2$.

    **(b).** The marginal distribution of $X_2$ is normal with mean $\mu_2$ and variance $\sigma_2^2$.

# Linear Discriminant Analysis when $p > 1$

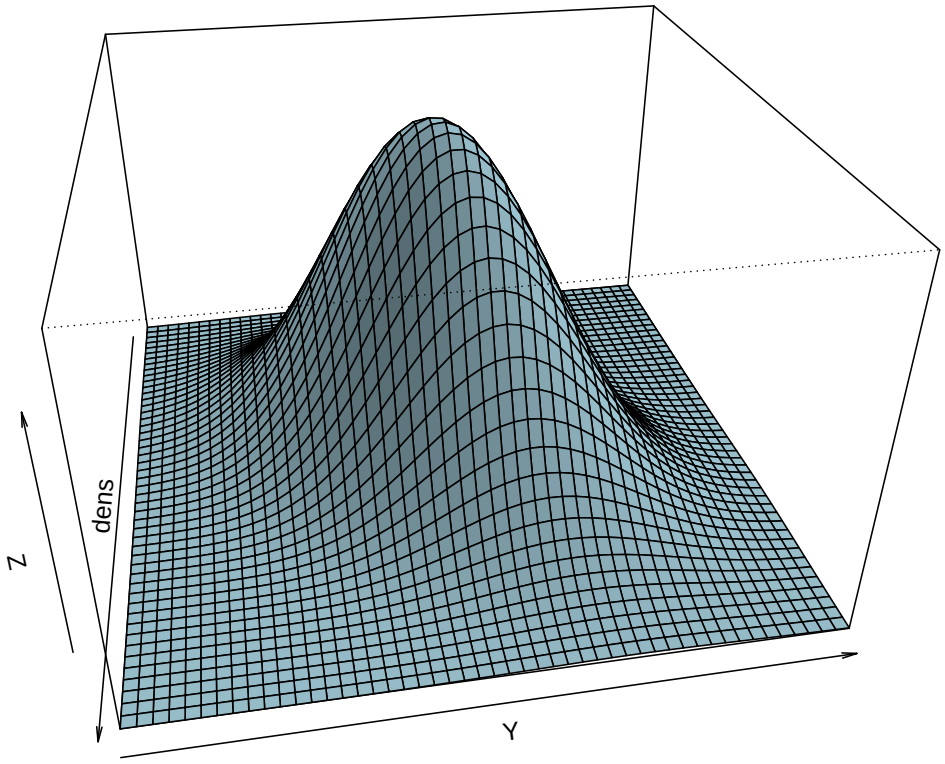
- Bivariate normal density



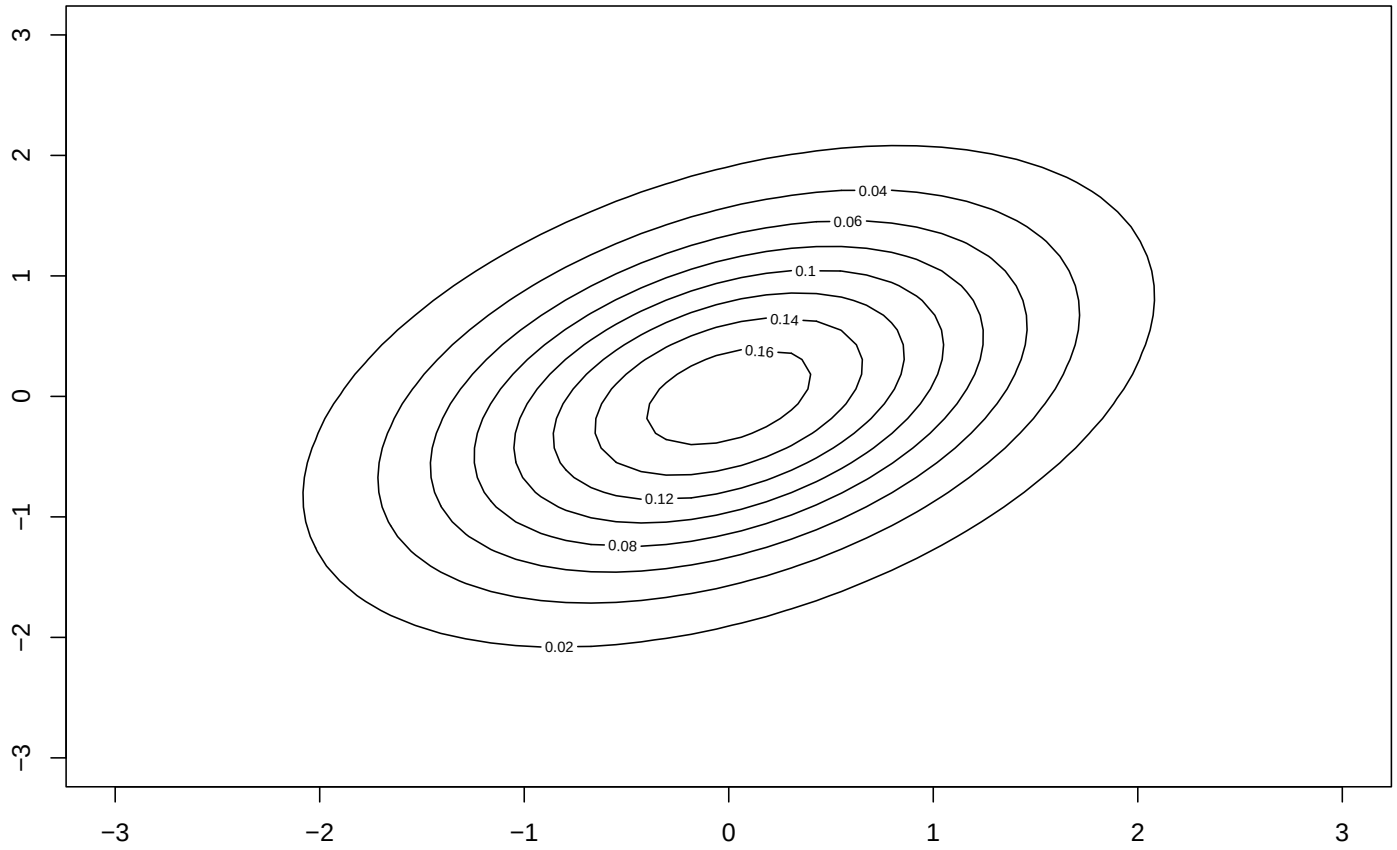
# Linear Discriminant Analysis when $p > 1$

- A bivariate normal with $\mu_1 = \mu_2 = 0, \sigma_1 = \sigma_2 = 1$ and $\rho = 0.4$



# Linear Discriminant Analysis when $p > 1$

- Contour plot

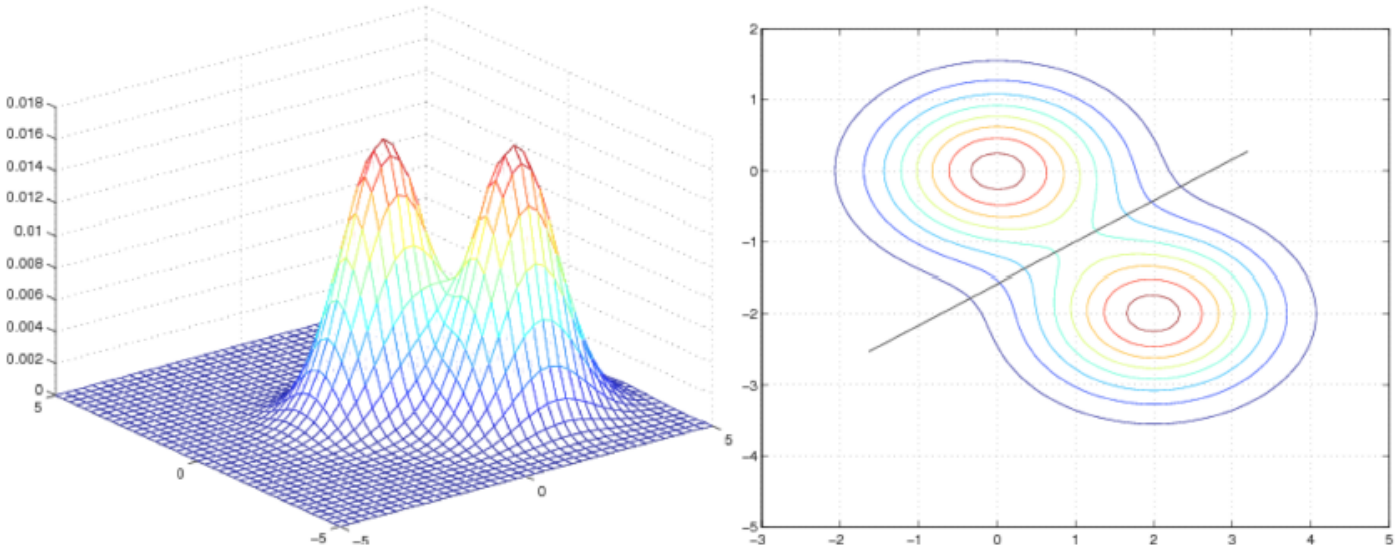


# Linear Discriminant Analysis when $p > 1$

- Discriminant function:

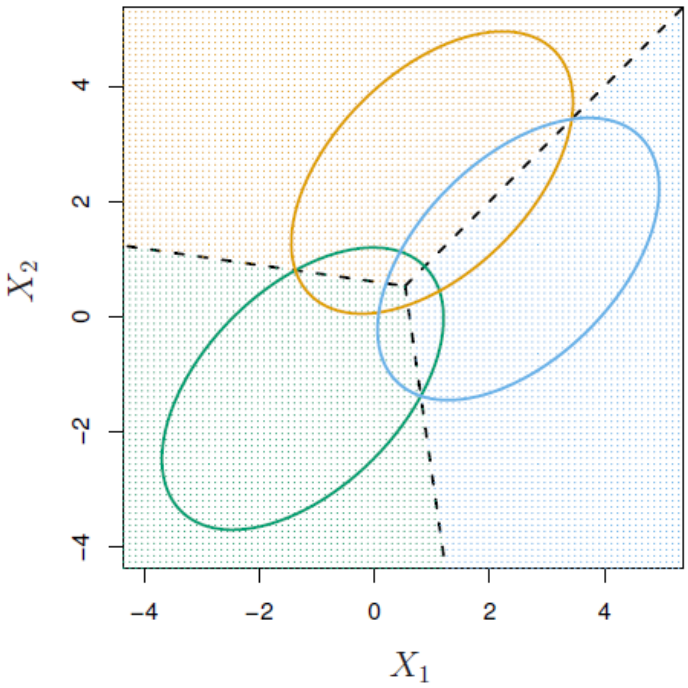
$\delta_k(\mathbf{x}) = \mathbf{x}^{'}\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^{'}\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_k + \log \pi_k$. Despite its complex form, it is a linear function of $X$:

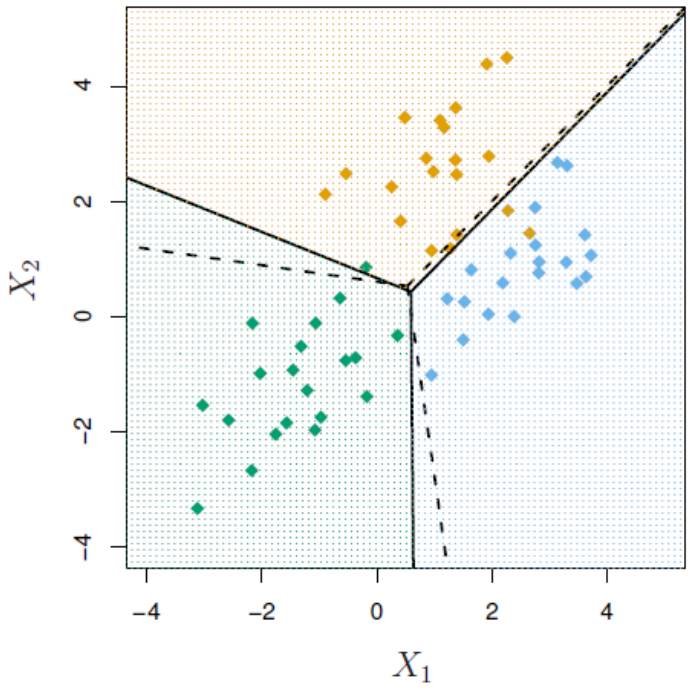
$$\delta_k(\mathbf{x}) = c_{k0} + c_{k1}x_1 + \cdots + c_{kp}x_p$$



# Linear Discriminant Analysis when $p > 1$

- Illustration: $p = 2$ and $K = 3$ classes
  - Here $\pi_1 = \pi_2 = \pi_3 = 1/3$
  - The dashed lines are the exact/true Bayes decision boundaries
  - The solid lines are LDA decision boundaries



# Linear Discriminant Analysis

- From $\delta_k(x)$ back to probabilities:

Once we have estimates $\delta_k(x)$, we can turn these into estimates for class probabilities:

$$\widehat{Pr}(Y = k|X = x) = \frac{e^{\delta_k(x)}}{\sum_{i=1}^{K} e^{\delta_i(x)}}$$

- So classifying to the largest $\delta_k(x)$ amounts to classifying to the class for which $\widehat{Pr}(Y = k|X = x)$ is largest.

- When $K = 2$, we classify to class 2 if $\widehat{Pr}(Y = 2|X = x) > 0.5$, else to class 1.

# Linear Discriminant Analysis

- Example: Consider the Stock Market Data `Smarket` again and split the data as training data and test data.

```
import pandas as pd
import numpy as np
df = pd.read_csv('../data/Smarket.csv')
X_train = df[df.Year<=2004][['Lag1','Lag2']]
y_train = df[df.Year<=2004]['Direction']
X_test = df[df.Year==2005][['Lag1','Lag2']]
y_test = df[df.Year==2005]['Direction']
```

# Linear Discriminant Analysis

- In Python, we can fit a LDA model using the `LinearDiscriminantAnalysis()` function, which is part of the `discriminant_analysis` module of the `sklearn` library.

- **Prior probabilities of groups**: the proportion of training observations in each group.
  - The LDA output indicates prior probabilities of $\hat{\pi}_1 = 0.492$ and $\hat{\pi}_2 = 0.508$; in other words, 49.2% of the training observations correspond to days during which the market went down.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
lda = LinearDiscriminantAnalysis()
model = lda.fit(X_train, y_train)

print(model.classes_)
```

```
## ['Down' 'Up']
```

```
print(model.priors_)
```

```
## [0.49198397 0.50801603]
```

# Linear Discriminant Analysis

- **Group means**: group center. Shows the mean of each variable in each group.

```
print(model.means_)
```

```
## [[ 0.04279022  0.03389409]
##  [-0.03954635 -0.03132544]]
```

- The above group means suggest that there is a tendency for the previous 2 days' returns to be negative on days when the market increases, and a tendency for the previous days' returns to be positive on days when the market declines.

# Linear Discriminant Analysis

- **Coefficients of linear discriminants**: Shows the linear combination of predictor variables that are used to form the LDA decision rule.
  - In this example, the response has only two categories
    - ★ LD1 = -0.0554*Lag1*-*0.0443*Lag2

```
print(model.coef_)
```

```
## [[-0.05544078 -0.0443452 ]]
```

# Linear Discriminant Analysis

- The `predict()` function returns a list of LDA's predictions about the movement of the market on the test data:

```
pred=model.predict(X_test)
print(confusion_matrix(pred, y_test).T)
```

```
## [[ 35  76]
##  [ 35 106]]
```

```
print(accuracy_score(pred, y_test))
```

```
## 0.5595238095238095
```

```
print(classification_report(y_test, pred, digits=3))
```

```
##               precision    recall  f1-score   support
##
##        Down      0.500     0.315     0.387       111
##          Up      0.582     0.752     0.656       141
##
##    accuracy                          0.560       252
##   macro avg      0.541     0.534     0.522       252
## weighted avg      0.546     0.560     0.538       252
```

# Linear Discriminant Analysis

- We can also get the predicted **posterior probabilities** using the `predict_proba()` function:

```
pred_p = model.predict_proba(X_test)
pred_p[0:4,]
```

```
## array([[0.49017925, 0.50982075],
##        [0.4792185 , 0.5207815 ],
##        [0.46681848, 0.53318152],
##        [0.47400107, 0.52599893]])
```

- Applying a 50% threshold to the posterior probabilities allows us to recreate the predictions contained in lda.pred$class.

```
# probabilities of going down
sum(pred_p[:,0]>=0.5)
```

```
## 70
```

```
sum(pred_p[:,0]<0.5)
```

```
## 182
```

# Linear Discriminant Analysis

- Notice that the above posterior probability output by the model corresponds to the probability that the market will decrease:
  - The second column are the probabilities that the market will increase

```
import numpy as np
np.round(pred_p[0:20,0],3)
```

```
## array([0.49 , 0.479, 0.467, 0.474, 0.493, 0.494, 0.495, 0.487, 0.
##        0.484, 0.491, 0.512, 0.49 , 0.471, 0.474, 0.48 , 0.494, 0.
##        0.498, 0.489])
```

```
pred[0:20]
```

```
## array(['Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up', 'Up'
##        'Down', 'Up', 'Up', 'Up', 'Up', 'Up', 'Down', 'Up', 'Up'],
##       dtype='<U4')
```

# Linear Discriminant Analysis

- If we wanted to use a posterior probability threshold other than 50% in order to make predictions, then we could easily do so. For instance, suppose that we wish to predict a market decrease only if we are very certain that the market will indeed decrease on that day—say, if the posterior probability is at least 80%.

```
sum(pred_p[:,0]>=0.8)
```

```
## 0
```

- No days in 2005 meet that threshold! In fact, the greatest posterior probability of decrease in all of 2005 was 52.02%.

```
max(pred_p[:,0])
```

```
## 0.5202349505356155
```

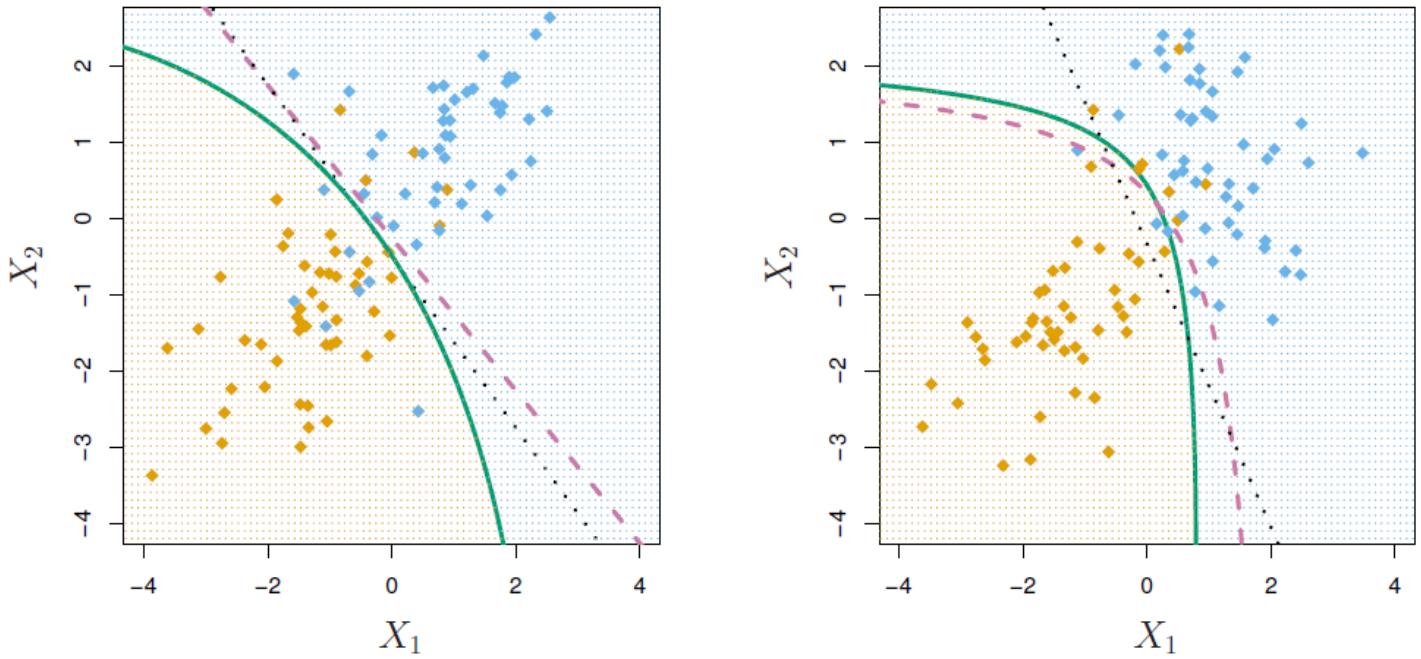
# Quadratic Discriminant Analysis

- LDA assumed that every class has the same variance/covariance
- However, LDA may perform poorly if this assumption is far from true
- QDA (Quadratic Discriminant Analysis) works identically as LDA except that it estimates separate variances/covariance for each class
- That is, $f_k(x) = Pr(Y = k|X = x)$ are Gaussian densities but with different variance-covariance matrix $\mathbf{\Sigma}_k$ in each class $k$, $k = 1, \ldots, K$.

# Quadratic Discriminant Analysis

- QDA results in non-linear decision boundaries (quadratic in fact)
- Discriminant function:

$$\delta_k(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})_k^{'}\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) + \log \pi_k - \frac{1}{2}\log|\boldsymbol{\Sigma}_k|$$



# Quadratic Discriminant Analysis

- We will now fit a QDA model to the `Smarket` data. QDA is implemented in `sklearn` using the `QuadraticDiscriminantAnalysis()` function, which is again part of the `discriminant_analysis` module. The syntax is identical to that of `LinearDiscriminantAnalysis()`.

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
qda = QuadraticDiscriminantAnalysis()
model2 = qda.fit(X_train, y_train)
```

# Quadratic Discriminant Analysis

- The output contains the group means. But it does not contain the coefficients of the linear discriminants, because the QDA classifier involves a quadratic, rather than a linear, function of the predictors.

```
print(model2.priors_)
```

```
## [0.49198397 0.50801603]
```

```
print(model2.means_)
```

```
## [[ 0.04279022  0.03389409]
##  [-0.03954635 -0.03132544]]
```

# Quadratic Discriminant Analysis

- The `predict()` function works in exactly the same fashion as for LDA.

```
pred2=model2.predict(X_test)
print(confusion_matrix(pred2, y_test).T)
```

```
## [[ 30  81]
##  [ 20 121]]
```

```
print(accuracy_score(pred2, y_test))
```

```
## 0.599206349206349
```

```
print(classification_report(y_test, pred2, digits=3))
```

```
##                precision    recall  f1-score   support
##
##         Down      0.600     0.270     0.373       111
##           Up      0.599     0.858     0.706       141
##
##     accuracy                          0.599       252
##    macro avg      0.600     0.564     0.539       252
## weighted avg      0.599     0.599     0.559       252
```

# Quadratic Discriminant Analysis

- Interestingly, the QDA predictions are accurate almost 60% of the time, even though the 2005 data was not used to fit the model. This level of accuracy is quite impressive for stock market data, which is known to be quite hard to model accurately. This suggests that the quadratic form assumed by QDA may capture the true relationship more accurately than the linear forms assumed by LDA and logistic regression.
- However, it is recommended to evaluate this method's performance on a larger test set before betting that this approach will consistently beat the market!

# Naive Bayes

- The method assumes **features are independent** in each class $k$, $k = 1, \ldots, K$.
- It is useful when $p$ is large, and so multivariate methods like QDA and even LDA break down.
- **Gaussian naive Bayes** assumes each $\mathbf{\Sigma}_k$ is diagonal (correlation is 0)

$$\begin{aligned}\delta_k(\mathbf{x}) &\propto \log\left[\pi_k \prod_{j=1}^{p} f_{kj}(x_j)\right] \\ &= -\frac{1}{2}\sum_{j=1}^{p}\left[\frac{(x_j-\mu_{kj})^2}{\sigma_{kj}^2} + \log\sigma_{kj}^2\right] + \log\pi_k\end{aligned}$$

- can use for mixed feature vectors (qualitative and quantitative). If $X_j$ is qualitative, replace $f_{kj}(x_j)$ with probability mass function (histogram) over discrete categories.
- Despite strong assumptions, naive Bayes often produces good classification results.

# Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
model3 = gnb.fit(X_train, y_train)
print(model3.classes_)
```

```
## ['Down' 'Up']
```

```
print(model3.class_prior_)
```

```
## [0.49198397 0.50801603]
```

```
print(model3.theta_) #mean of each feature per class
```

```
## [[ 0.04279022  0.03389409]
##  [-0.03954635 -0.03132544]]
```

# Naive Bayes

```
pred3=model3.predict(X_test)
print(confusion_matrix(pred3, y_test).T)
```

```
## [[ 29  82]
##  [ 20 121]]
```

```
print(accuracy_score(pred3, y_test))
```

```
## 0.5952380952380952
```

```
print(classification_report(y_test, pred3, digits=3))
```

```
##               precision    recall  f1-score   support
##
##         Down      0.592     0.261     0.362       111
##           Up      0.596     0.858     0.703       141
##
##     accuracy                          0.595       252
##    macro avg      0.594     0.560     0.533       252
## weighted avg      0.594     0.595     0.553       252
```

# DA vs Logistic Regression

- Discriminant Analysis model can actually be rewritten as multinomial logistic models:

Beginning with

$$p_k(\mathbf{x}) = Pr(Y = k|\mathbf{X} = \mathbf{x}) = \frac{\pi_k f_k(\mathbf{x})}{\sum_{i=1}^{K} \pi_i f_i(\mathbf{x})},$$

and

$$f_k(\mathbf{x}) \propto \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})_k^{'} \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right]$$

substituting and simplifying we get

$$P(Y = k|\mathbf{x}) = \frac{e^{\eta_k}}{\sum_i e^{\eta_i}}$$

where $\eta_k = \beta_0 + \mathbf{x}^{'}\boldsymbol{\beta} + \mathbf{x}^{'}\boldsymbol{\Sigma}_k^{-1}\mathbf{x}$.

# DA vs Logistic Regression

- This is just a multinomial logistic model with quadratic terms and interactions.
- In particular for LDA (where $\mathbf{\Sigma}_k = \mathbf{\Sigma}$ is pooled) we have cancellation and get

$$\eta_k = \beta_0 + \mathbf{x}^{'}\boldsymbol{\beta} = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$$

  which is simply a linear logistic model.
- The difference is in how the parameters are estimated.
- Logistic regression uses the conditional likelihood based on $Pr(Y|\mathbf{x})$ (known as *discriminative learning).*
- *LDA uses the full likelihood based on* $Pr(\mathbf{X}, Y)$ *(known as generative learning).*
- *Despite these differences, in practice the results are often very similar.*

# Summary

- Logistic regression is very popular for classification, especially when $K = 2$.
- LDA is useful when $n$ is small, or the classes are well separated, and Gaussian assumptions are reasonable. Also when $K > 2$.
- Both Logistic Regression and LDA produce linear boundaries. LDA would do better than Logistic Regression if the assumption of normality hold, otherwise logistic regression can outperform LDA
- KNN is completely non-parametric: No assumptions are made about the shape of the decision boundary.
- We can expect KNN to dominate both LDA and Logistic Regression when the decision boundary is highly non-linear. But KNN does not tell us which features/predictors are important (no table of coefficients)

# Summary

- Naive Bayes is useful when $p$ is very large.
- QDA is a compromise between non-parametric KNN method and the linear LDA and logistic regression
- If the true decision boundary is:
  - Linear: LDA and Logistic outperforms
  - Moderately Non-linear: QDA outperforms
  - More complicated: KNN is superior

# Stock Market Data by Logistic Regression

- Stock Market Data by Logistic Regression

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
train = df[df.Year<=2004][['Lag1','Lag2','Direction']]
#Note that Direction is non-numeric
Log_model = smf.glm(formula = 'Direction~Lag1+Lag2',
data = train,family = sm.families.Binomial())
model4 = Log_model.fit()
```

# Stock Market Data by Logistic Regression

```
print(model4.summary())
```

```
##                          Generalized Linear Model Regression Results
## ==============================================================================
## Dep. Variable:     ['Direction[Down]', 'Direction[Up]']   No. Observations:
## Model:                                              GLM   Df Residuals:
## Model Family:                                  Binomial   Df Model:
## Link Function:                                    Logit   Scale:
## Method:                                            IRLS   Log-Likelihood:
## Date:                                 Tue, 18 Apr 2023   Deviance:
## Time:                                         14:43:56   Pearson chi2:
## No. Iterations:                                      4   Pseudo R-squ. (CS):
## Covariance Type:                             nonrobust
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## Intercept     -0.0322      0.063     -0.508      0.611      -0.156       0.092
## Lag1           0.0556      0.052      1.076      0.282      -0.046       0.157
## Lag2           0.0445      0.052      0.861      0.389      -0.057       0.146
## ==============================================================================
```

# Stock Market Data by Logistic Regression

- The above `glm` function models probabilities of the response `Down`
  - You my try to run the following code to verify this

```
df['direction'] = [1 if x =="Down"
else 0 for x in df['Direction']]
train = df[df.Year<=2004][['Lag1','Lag2','direction']]
Log_model = smf.glm(formula = 'direction~Lag1+Lag2',
data = train,family = sm.families.Binomial())
model4 = Log_model.fit()
model4.summary()
```

# Stock Market Data by Logistic Regression

```
import numpy as np
predictions = model4.predict(X_test)
pred4=[ "Down" if x >= 0.5 else "Up" for x in predictions]
print(confusion_matrix(pred4, y_test).T)
```

```
## [[ 35  76]
##  [ 35 106]]
```

```
print(accuracy_score(pred4, y_test))
```

```
## 0.5595238095238095
```

```
print(classification_report(y_test, pred4, digits=3))
```

```
##               precision    recall  f1-score   support
##
##         Down      0.500     0.315     0.387       111
##           Up      0.582     0.752     0.656       141
##
##     accuracy                          0.560       252
##    macro avg      0.541     0.534     0.522       252
## weighted avg      0.546     0.560     0.538       252
```

```
library(ISLR);
```

# Model selections with CV

- We did not discuss model/feature selections for the classification methods:
  - KNN (what $k$ should be used)
  - LDA
  - QDA
  - Naive Bayes
  - Logistic Regression

# Model selections with CV

- Suppose we have training data `X_train` and `y_train`
- For the kNN method, we can use function `cross_val_score` to get the cross-validatation score for a specific $k$ value.

```
import numpy as np
from sklearn.model_selection import cross_val_score
knn_cv = KNeighborsClassifier(n_neighbors=k)
cv_scores = cross_val_score(knn_cv, X_train, y_train,
scoring ='accuracy', cv=10)
np.mean(cv_scores)
```

- For a grid of $k$ values $k = 1, 2, 3, \cdots$, we use a `for` loop and choose the model with larges accuacy score.

# Model selections with CV

- Function `SelectKBest` in `sklearn.feature_selection` can be used for feature selection in logistic regression models.

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import cross_val_score

logreg = LogisticRegression()

feature_selector = SelectKBest(k=p)
# Choose the top k features; k=p all features included

scores = cross_val_score(logreg, feature_selector.fit_transform(X_train, y_train),
y_train, scoring ='accuracy',cv=10) #10-fold cv

np.mean(cv_scores)
```

- Again, for a grid of $k$ values, we use a `for` loop and choose the model with largest accuracy score.
- Similarly, we can conduct feature selection for LDA, QDA and Naive Bayes.

# License