Machine Learning 2023/2024 (2nd semester)

Master in Electrical and Computer Engineering

Department of Electrical and Computer Engineering

A. Pedro Aguiar (pedro.aguiar@fe.up.pt), **Aníbal Matos** (anibal@fe.up.pt), **Andry Pinto** (amgp@fe.up.pt), **Daniel Campos** (dfcampos@fe.up.pt), **Maria Inês Pereira** (maria.ines@fe.up.pt)

FEUP, April 2024

Notebook #06: Classification (part 1)

Bayesian Classification

Bayes' theorem

It is used to calculate "conditional probabilities", which is very useful when the probability of a given event to occur is afected by previous events. This way, we have the general case:

$$P(y \ x) = \frac{P(x \ y)P(y)}{P(x)}$$

where $P(y\ \ x)$ is the probability of event y given event x has occurred (posterior probability), $P(x\ \ y)$ is the probability of event x given event y has occurred (likelihood function), $P(y)$ as the prior probability of event y (what we already know about event y), and $P(x)$ as the prior probability of event x (evidence function which acts as a normalization constant).

Bayesian classification rule

Given a set of M classes ω_i (the classes for the classification process), for $i = 1, \ldots, M$, and the posterior probabilities $P(\omega_i\ \ x)$:

$$\text{Assign } x \text{ to } \omega_i = \arg\max_{\omega_j} P(\omega_j\ \ x), \quad j = 1, \ldots, M$$

Note that from Bayes' theorem:

$$P(\omega_j\ \ x) = \frac{p(x\ \ \omega_j)P(\omega_j)}{p(x)}, \quad j = 1, \ldots, M$$

Then, the Bayesian classification rule becomes: ($p(x)$ can be ignored because it does not depend on ω_j)

$$\text{Assign } x \text{ to } \omega_i = \arg\max_{\omega_j} p(x\ \ \omega_j)P(\omega_j), \quad j = 1, \ldots, M$$

Misclassification error

Let $\mathcal{R}_i$ be the region where we decide for class ω_i $(i = 1, \ldots, M)$. The probability of making a classification error is

$$P_e = \sum_{i=1}^{M} P(\omega_i) \int_{U \setminus \mathcal{R}_i} p(x\ \ \omega_i)dx$$

For the 2-class problem, this probability is

$$P_e = P(\omega_2) \int_{\mathcal{R}_1} p(x\ \ \omega_2)dx + P(\omega_1) \int_{\mathcal{R}_2} p(x\ \ \omega_1)dx$$

Minimizing the expected risk

In some cases, misclassification errors might not be equally significant. For such cases we can consider relative weights on the errors according to their significance and define an overall risk (or loss). The risk associated with class ω_k is defined as

$$r_k = \sum_{i=1}^{M} \lambda_{ki} \sum_{\mathcal{R}_i} p(x \ \omega_k) dx$$

where λ_{ki} is the relative weight associated with committing an error by assigning a pattern from class ω_k to class ω_i . Note that $\lambda_{kk} = 0$ (since it corresponds to correct decisions). The average risk is then

$$r = \sum_{k=1}^{M} P(\omega_k) r_k = \sum_{i=1}^{M} \int_{\mathcal{R}_i} \left(\sum_{k=1}^{M} \lambda_{ki} P(\omega_k) p(x \ \omega_k) \right) dx$$

The average risk is minimized if the space is partitioned by selecting $\mathcal{R}_i$ such that the integrals in the above expression are minimized. Such is achieved by adopting the rule

$$\text{Assign } x \text{ to } \omega_i = \arg \min_{\omega_j} \sum_{k=1}^{M} \lambda_{kj} P(\omega_k) p(x \ \omega_k), \quad j = 1, \ldots, M$$

For the 2-class case, this reduces to

$$\text{Assign } x \text{ to } \omega_1 \text{ if: } \lambda_{21} P(\omega_2) p(x \ \omega_2) < \lambda_{12} P(\omega_1) p(x \ \omega_1)$$
$$\text{Assign } x \text{ to } \omega_2 \text{ if: } \lambda_{12} P(\omega_1) p(x \ \omega_1) < \lambda_{21} P(\omega_2) p(x \ \omega_2)$$

Minimum distance classifier

Under the following assumptions:

i) Data follows a Gaussian distribution in each one of the classes;

ii) All classes are equiprobable; and

iii) Covariance is equal for all classes (Σ);

the Bayesian classification rule is equivalent to

$$\text{Assign } x \text{ to class } \omega_i : i = \arg\min_j \| x - \mu_j \|^2_{\Sigma^{-1}}, \quad j = 1, \ldots, M$$

where $\| z \|_{\Sigma^{-1}} = \sqrt{z^T \Sigma^{-1} z}$ is the Mahalanobis distance.

This means that classifying x consists in selecting the class whose mean minimizes the **Mahalanobis distance** to x. When the (common) covariance has the form $\Sigma = \sigma^2 I$ (all features share the same variance), this is equivalent to minimizing the Euclidean distance.

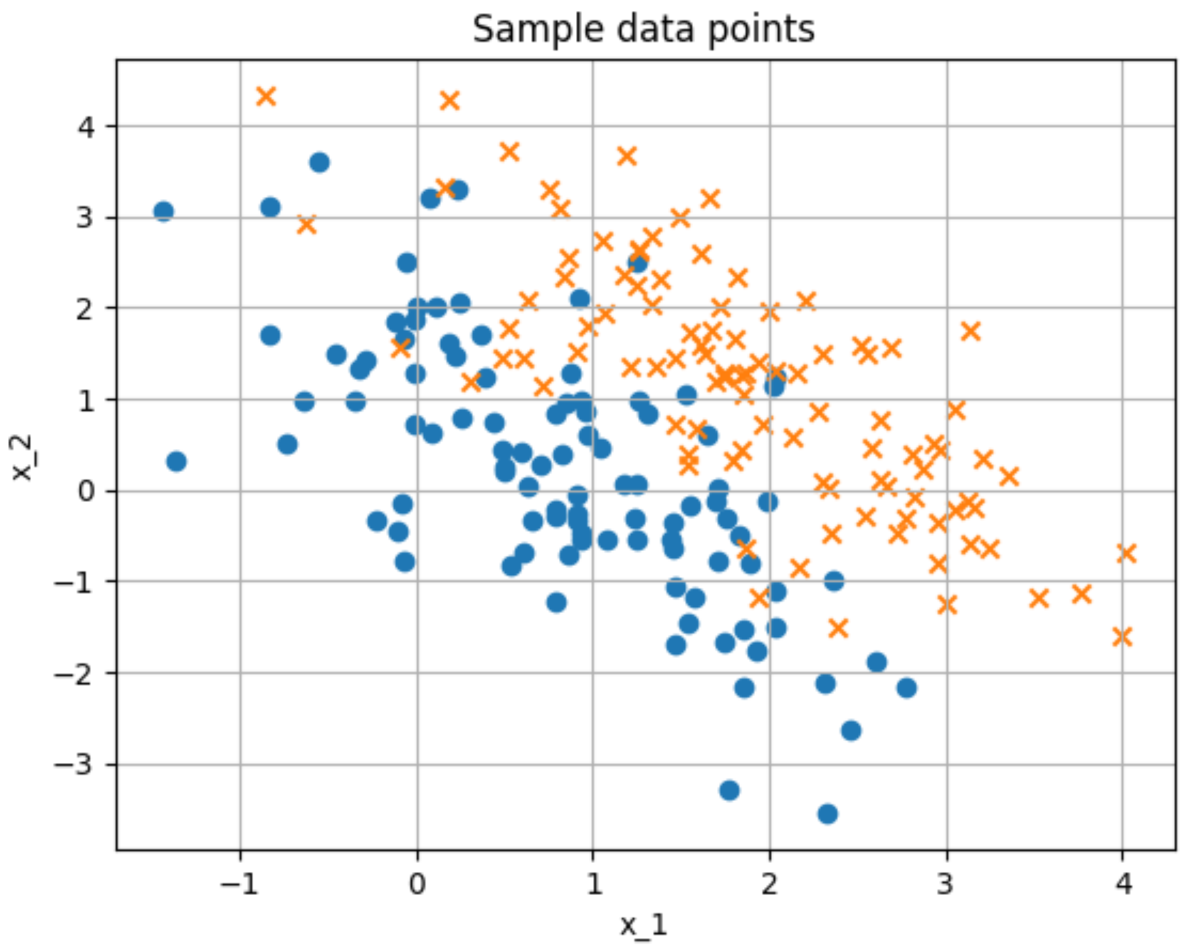
Activity 1

Consider a two-dimensional class problem with two classes (ω_1 and ω_2), characterized by Gaussian distributions with means $\mu_1 = (1, 0)^\top$ and $\mu_2 = (2, 1)^\top$, and covariances $\Sigma_1 = \Sigma_2 = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$, respectively. Assume the classes are equiprobable.

1.1 Obtain a dataset consisting of 100 points from each class. Plot the data.

```
# Activity 1.1
import numpy as np
import matplotlib.pyplot as plt

# to make sure we have always the same data
np.random.seed(10)
```

```
# Distribution for class 1
mu_1 = np.array([1,0])
sigma_1 = np.array([[1,-1],[-1,2]])

# Distribution for class 2
mu_2 = np.array([2,1])
sigma_2 = np.array([[1,-1],[-1,2]])

# Sample from class distributions
size_1 = 100
size_2 = 100
X_1 = np.random.multivariate_normal(mu_1, sigma_1, size=size_1)
X_2 = np.random.multivariate_normal(mu_2, sigma_2, size=size_2)

# plot data points
plt.title('Sample data points')
plt.xlabel('x_1')
plt.ylabel('x_2')
# Class 1 (o)
plt.scatter(X_1[:,0], X_1[:,1], marker='o')
# Class 2 (x)
plt.scatter(X_2[:,0], X_2[:,1], marker='x')

# A diffent plotting scope
# plt.xlim([-4, 4])
# plt.xlim([-4, 4])

plt.plot()
plt.grid()
plt.show()
```



1.2 Assign each point of the dataset to either ω_1 or ω_2, according to the Bayes decision rule. Estimate the classification probability error and plot the classification regions and the points.

```
# Activity 1.2 (to be completed)
```

```
# multivariate gaussian distribution
def gaussian(x, mean, cov):
    n = mean.size
    d = x-mean
    np.reshape(d,[n,1])
    exp_term = -0.5 * np.transpose(d) @ np.linalg.inv(cov) @ d
    f_term = 1.0/(np.sqrt( (2*np.pi)**n * np.linalg.det(cov)))
    pdf = f_term * np.exp(exp_term)
    return pdf

size_total = size_1+size_2
label_1 =   np.ones(size_1)
label_2 = 2*np.ones(size_2)

# put all data together
X = np.concatenate([X_1, X_2], axis=0)
label = np.concatenate([label_1, label_2], axis=0)

# A priori probabilities
# to be completed
#P_class1 = ...
#P_class2 = ...

# classify each data point (Bayesian classification)
classification = []
for i in range(size_total):
  # to be completed... (compute conditional probabilities)
  #prob_1 =
  #prob_2 =
  # to be completed (classify)
  #if ... :
```

```
  #  classification.append(1)
  #else :
  #  classification.append(2)

# obtain classification probability error
Pe=0
for i in range(size_total) :
  # to be completed
  #if ... :
  #  Pe = Pe + 1
Pe = Pe/size_total

print('Classification probability error:', Pe)

# plot classification regions and points

# generate a large number of points to create the regions
x_min, x_max = X[:,0].min() - .5, X[:,0].max() + .5
y_min, y_max = X[:,1].min() - .5, X[:,1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02), np.arange(y_min, y_max, .02))
X_plot = np.c_[xx.ravel(), yy.ravel()]

# classify all points from X_plot data
prediction = []
for i in range(X_plot.shape[0]):
  # to be completed... (compute conditional probabilities)
  #prob_1 =
  #prob_2 =
  # to be completed (classify)
  #if ... :
  #  prediction.append(1)
  #else :
```

```
  #  prediction.append(2)

prediction = np.array(prediction)
prediction = prediction.reshape(xx.shape)

plt.figure(1, figsize=(5, 4))
plt.pcolormesh(xx, yy, prediction, cmap=plt.cm.Paired)

plt.scatter(X[:, 0], X[:, 1], c=label, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('x_1')
plt.ylabel('x_2')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```

```
  File "<ipython-input-10-10432d9f5867>", line 29
    Pe=0
    ^
IndentationError: expected an indented block
```

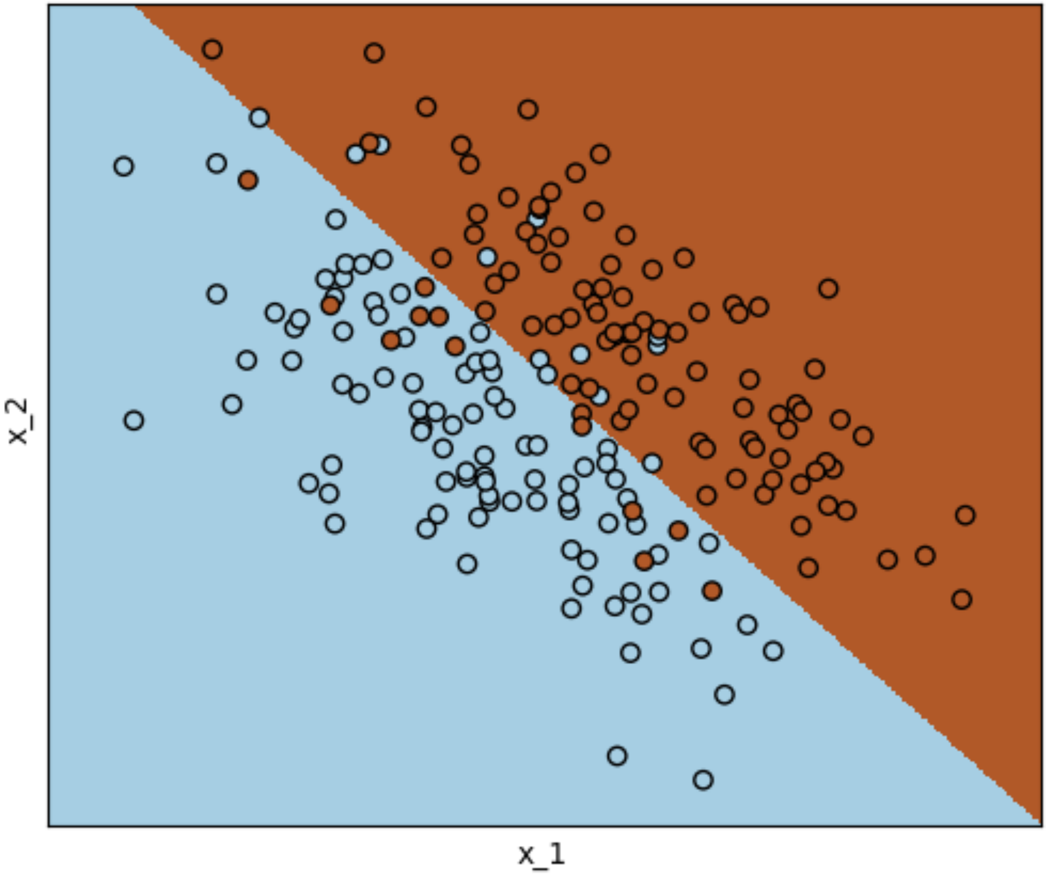
```
# Activity 1.2 (solution)

# multivariate gaussian distribution
def gaussian(x, mean, cov):
    n = mean.size
    d = x-mean
```

```
    np.reshape(d,[n,1])
    exp_term = -0.5 * np.transpose(d) @ np.linalg.inv(cov) @ d
    f_term = 1.0/(np.sqrt( (2*np.pi)**n * np.linalg.det(cov)))
    pdf = f_term * np.exp(exp_term)
    return pdf

size_total = size_1+size_2
label_1 =   np.ones(size_1)
label_2 = 2*np.ones(size_2)

# put all data together
X = np.concatenate([X_1, X_2], axis=0)
label = np.concatenate([label_1, label_2], axis=0)

# A priori probabilities
P_class1 = size_1/size_total
P_class2 = size_2/size_total

# classify each data point (Bayesian classification)
classification = []
for i in range(size_total):
  prob_1 = gaussian(X[i],mu_1,sigma_1) * P_class1
  prob_2 = gaussian(X[i],mu_2,sigma_2) * P_class2
  if prob_1 > prob_2 :
    classification.append(1)
  else :
    classification.append(2)

# obtain classification probability error
Pe=0
for i in range(size_total) :
  if classification[i] != label[i] :
```

```
    Pe = Pe + 1

Pe = Pe/size_total

print('Classification probability error:', Pe)

# plot classification regions and points

# generate a large number of points to create the regions
x_min, x_max = X[:,0].min() - .5, X[:,0].max() + .5
y_min, y_max = X[:,1].min() - .5, X[:,1].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02), np.arange(y_min, y_max, .02))
X_plot = np.c_[xx.ravel(), yy.ravel()]

# classify all points
prediction = []
for i in range(X_plot.shape[0]):
  prob_1 = gaussian(X_plot[i],mu_1,sigma_1) * P_class1
  prob_2 = gaussian(X_plot[i],mu_2,sigma_2) * P_class2
  if prob_1 > prob_2 :
    prediction.append(1)
  else:
    prediction.append(2)

prediction = np.array(prediction)
prediction = prediction.reshape(xx.shape)

plt.figure(1, figsize=(6, 5))
plt.pcolormesh(xx, yy, prediction, cmap=plt.cm.Paired)

plt.scatter(X[:, 0], X[:, 1], c=label, edgecolors='k', cmap=plt.cm.Paired)
#print (label)
```

```
plt.xlabel('x_1')
plt.ylabel('x_2')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```

```
Classification probability error: 0.125
```



1.3 Consider the loss matrix $L = \begin{bmatrix} 0 & 1 \\ 0.05 & 0 \end{bmatrix}$. Assign each point of the dataset to ω_1 or ω_2 according to the average risk minimization rule. Plot the points with different colors, depending on the class they are assigned to. Estimate the average risk for this loss matrix.

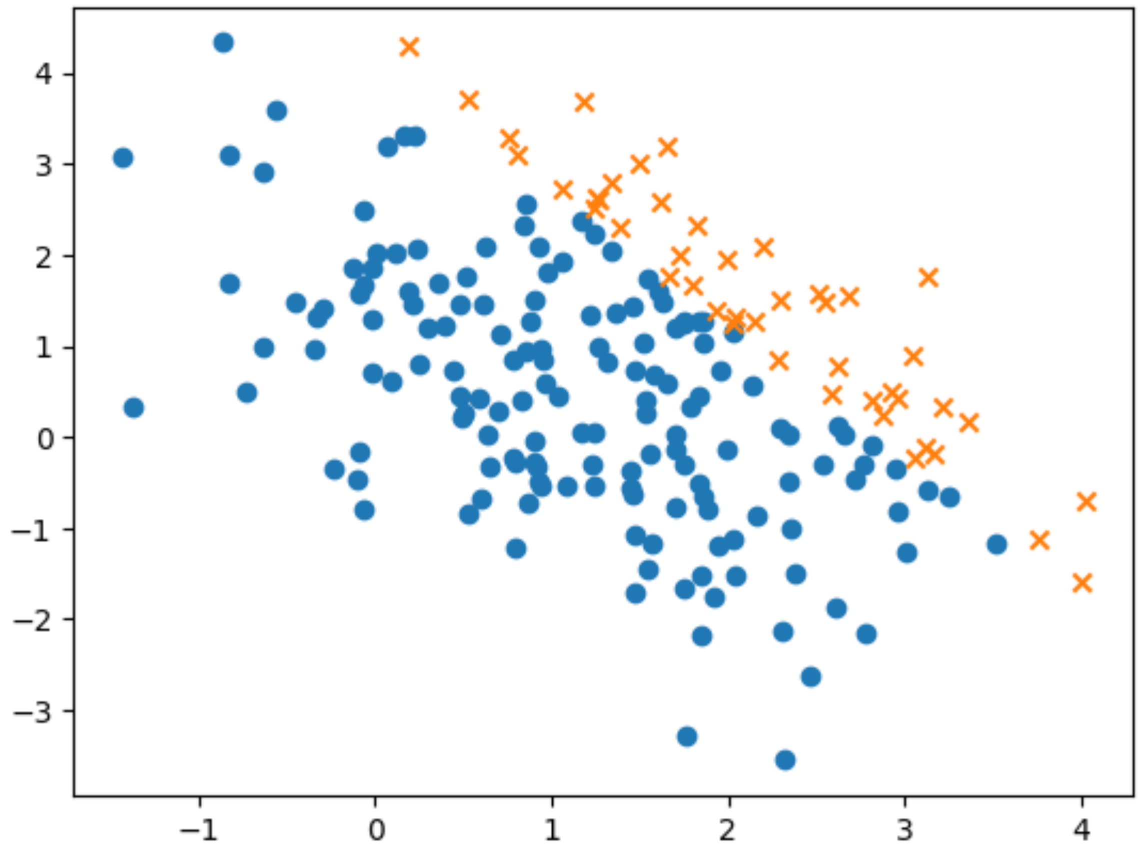
```
# Activity 1.3 (to be completed)
```

```
# loss matrix entries
L12 = 1
L21 = 0.05

# classifify each data point (average risk minimization)
class_risk = []
for i in range(size_total):
  # to be completed
  #prob_1 = ...
  #prob_2 = ...
  #if ... :
  #  class_risk.append(1)
  #else :
  #  class_risk.append(2)

class_risk = np.array(class_risk)

plt.figure()
plt.scatter(X[np.where(class_risk==1),0], X[np.where(class_risk==1),1], marker='o')
plt.scatter(X[np.where(class_risk==2),0], X[np.where(class_risk==2),1], marker='x')
plt.plot()
plt.show()

# determine average risk
Ar = 0
for i in range(size_total) :
  if class_risk[i]!=label[i] :
    # to be completed
    #if ... :
    #  Ar = Ar + L12
    #else :
```

```
    #  Ar = Ar + L21

Ar = Ar/size_total
print('Average risk:', Ar)
```

```
# Activity 1.3 (solution)

# loss matrix entries
L12 = 1
L21 = 0.05

# classififty each data point (average risk minimization)
class_risk = []
for i in range(size_total):
  prob_1 = gaussian(X[i],mu_1,sigma_1) * P_class1
  prob_2 = gaussian(X[i],mu_2,sigma_2) * P_class2
  if L12*prob_1 > L21*prob_2:
    class_risk.append(1)
  else:
    class_risk.append(2)

class_risk = np.array(class_risk)

plt.figure()
plt.scatter(X[np.where(class_risk==1),0], X[np.where(class_risk==1),1], marker='o')
plt.scatter(X[np.where(class_risk==2),0], X[np.where(class_risk==2),1], marker='x')
plt.plot()
plt.show()

# determine average risk
Ar = 0
```

```
for i in range(size_total) :
  if class_risk[i]!=label[i] :
    if label[i]==1 :
      Ar = Ar + L12
    else :
      Ar = Ar + L21

Ar = Ar/size_total
print('Average risk:', Ar)
```



```
Average risk: 0.024249999999999995
```

1.4 Using the same data, verify that in this case the Bayesian classification is in fact equivalent to the minimization of the Mahalanobis distance.

```
# Activity 1.4 (to be completed)

# classify each data point (Mahalanobis distance)
Sigma_inv = np.linalg.inv(sigma_1)
classification_dM = []
for i in range(size_total):
  diff1 = X[i]-mu_1
  diff1 = np.transpose([diff1])
  diff2 = X[i]-mu_2
  diff2 = np.transpose([diff2])
  # to be completed
  #dM1_squared = ...
  #dM2_squared = ...
  #if ... :
  #  classification_dM.append(1)
  #else :
  #  classification_dM.append(2)

mclass = 0
if i in range(size_total) :
  if classification[i] != classification_dM[i] :
      print(i,' was not assigned to the same class')
      mclass += 1

if mclass == 0 : print('all points were classified in the same way')
```

```
# Activity 1.4 (solution)

# classify each data point (Mahalanobis distance)
Sigma_inv = np.linalg.inv(sigma_1)
classification_dM = []
for i in range(size_total):
  diff1 = X[i]-mu_1
  diff1 = np.transpose([diff1])
  diff2 = X[i]-mu_2
  diff2 = np.transpose([diff2])
  dM1_squared = np.transpose(diff1) @ Sigma_inv @ diff1
  dM2_squared = np.transpose(diff2) @ Sigma_inv @ diff2
  if dM1_squared < dM2_squared :
    classification_dM.append(1)
  else :
    classification_dM.append(2)

mclass = 0
if i in range(size_total) :
  if classification[i] != classification_dM[i] :
      print(i,' was not assigned to the same class')
      mclass += 1

if mclass == 0 : print('all points were classified in the same way')
```

```
all points were classified in the same way
```

Naive Bayes classifier

The basic assumption now is that features (components of the feature vector) are statistically independent. This means the joint distribution is the product of the marginal distributions of the features, that is

$$p(x \ \omega_i) = \prod_{k=1}^{l} p(x_k \ \omega_i), \quad i = 1, \ldots, M$$

where we assume there are l features.

The classification rule is then

$$\text{Assign } x \text{ to } \omega_i = \arg\max_{\omega_j} P(\omega_j) \prod_{k=1}^{l} p(x_k \ \omega_j), \quad j = 1, \ldots, M$$

Activity 2

Consider a set of data with two features (X_1 and X_2) from a representative data set of points of two classes (ω_1 and ω_2).

```
!wget -O dataset.csv.zip https://www.dropbox.com/s/evpwqery7uleqw1/data-set.csv.zip?dl=0 --quiet
!unzip dataset.csv.zip -d.
```

```
Archive:  dataset.csv.zip
  inflating: ./data-set.csv
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# multivariate gaussian distribution
def gaussian(x, mean, cov):
    n = mean.size
    d = x-mean
    np.reshape(d,[n,1])
    exp_term = -0.5 * np.transpose(d) @ np.linalg.inv(cov) @ d
    f_term = 1.0/(np.sqrt( (2*np.pi)**n * np.linalg.det(cov)))
    pdf = f_term * np.exp(exp_term)
    return pdf

# 1d normal distribution
def gaussian1d(x, mean, var):
    pdf = 1/np.sqrt(2*np.pi*var)*np.exp(-0.5*(x-mean)**2/var)
    return pdf

# read data from file
df = pd.read_csv('data-set.csv')
npoints = df.values.shape[0]
```

2.1 Plot the data from the data set. Estimate the mean and covariance of the distributions within each class. Also estimate the probability of occurrence of each class.

Note that

$$\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} x_n, \quad \hat{\Sigma} = \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \hat{\mu})(x_n - \hat{\mu})^\top$$

```
# Activity 2.1 (to be completed)

# plot data
classData = df.values[:,2]
plt.figure()
plt.scatter(df.values[np.where(classData==1),0], df.values[np.where(classData==1),1], marker='o')
plt.scatter(df.values[np.where(classData==2),0], df.values[np.where(classData==2),1], marker='x')
plt.plot()
plt.show()



# estimate mean and probability of each class
sum_X_class1 = np.array([0,0])
sum_X_class2 = np.array([0,0])
num_class1 = 0
num_class2 = 0

# cycle through all points
for i in range(npoints) :
  # to be completed
  if df.values[i,2] == 1 :
    #sum_X_class1 = ...
```

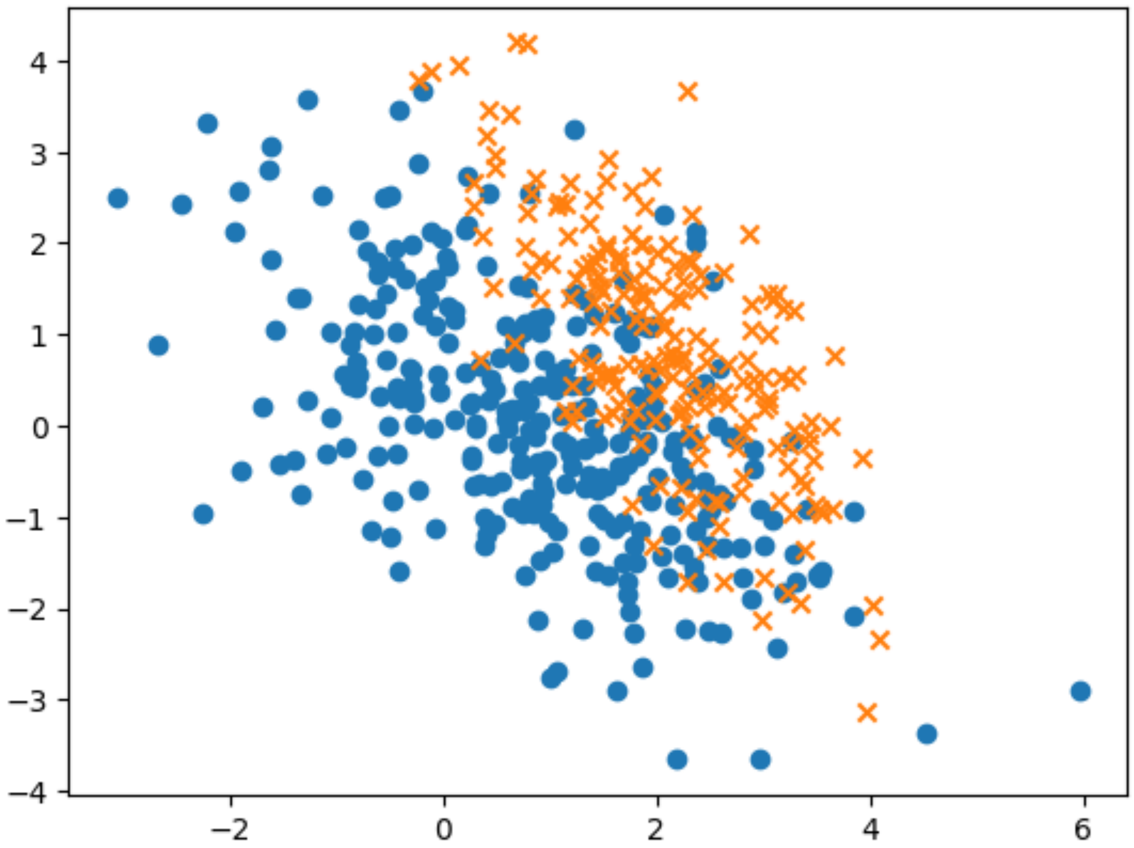
```
    #num_class1 = ...
  else :
    #sum_X_class2 = ...
    #num_class2 = ...

# compute probability
# to be completed
#P_class1 =  ...
#P_class2 =  ...

# compute mean
# to be completed
#mu_class1 = ...
#mu_class2 = ...

# display probablities and means
print('P1:',P_class1)
print('P2:',P_class2)
print('mu1:',mu_class1)
print('mu2:',mu_class2)

# estimate covariance of each class
sum_XXT_class1 = np.array([[0,0],[0,0]])
sum_XXT_class2 = np.array([[0,0],[0,0]])
for i in range(npoints) :
  # to be completed
  if df.values[i,2] == 1 :
    #sum_XXT_class1 = ...
  else :
    #sum_XXT_class2 = ...

# compute convariance
```

```
# to be completed
#sigma_class1 = ...
#sigma_class2 = ...

print('Sigma1:',sigma_class1)
print('Sigma2:',sigma_class2)
```

```
# Activity 2.1 (solution)

# plot data
classData = df.values[:,2]
plt.figure()
plt.scatter(df.values[np.where(classData==1),0], df.values[np.where(classData==1),1], marker='o')
plt.scatter(df.values[np.where(classData==2),0], df.values[np.where(classData==2),1], marker='x')
plt.plot()
plt.show()


# estimate mean and probability of each class
sum_X_class1 = np.array([0,0])
sum_X_class2 = np.array([0,0])
num_class1 = 0
num_class2 = 0

# cycle through all points
for i in range(npoints) :
    if df.values[i,2] == 1 :
        sum_X_class1 = sum_X_class1 + np.array(df.values[i,0:2])
        num_class1 += 1
    else :
        sum_X_class2 = sum_X_class2 + np.array(df.values[i,0:2])
```

```
        num_class2 += 1

# compute probability
P_class1 =  num_class1 / npoints
P_class2 =  num_class2 / npoints

# compute mean
mu_class1 = sum_X_class1 / num_class1
mu_class2 = sum_X_class2 / num_class2

# display probablities and means
print('P1:',P_class1)
print('P2:',P_class2)
print('mu1:',mu_class1)
print('mu2:',mu_class2)

# estimate covariance of each class
sum_XXT_class1 = np.array([[0,0],[0,0]])
sum_XXT_class2 = np.array([[0,0],[0,0]])
for i in range(npoints) :
    if df.values[i,2] == 1 :
        d = np.array(df.values[i,0:2])-mu_class1
        d = np.transpose([d])
        sum_XXT_class1 = sum_XXT_class1 +  d @ np.transpose(d)
    else :
        d = np.array(df.values[i,0:2])-mu_class2
        d = np.transpose([d])
        sum_XXT_class2 = sum_XXT_class2 + d @ np.transpose(d)

sigma_class1 = 1/(num_class1-1) * sum_XXT_class1
sigma_class2 = 1/(num_class2-1) * sum_XXT_class2
```

```
print('Sigma1:',sigma_class1)
print('Sigma2:',sigma_class2)
```




```
P1: 0.6
P2: 0.4
mu1: [0.88839063 0.05565378]
mu2: [2.08024385 0.8545214 ]
Sigma1: [[ 1.87290892 -1.01085984]
 [-1.01085984  1.81633253]]
Sigma2: [[ 0.80589441 -0.81234663]
 [-0.81234663  1.73189176]]
```

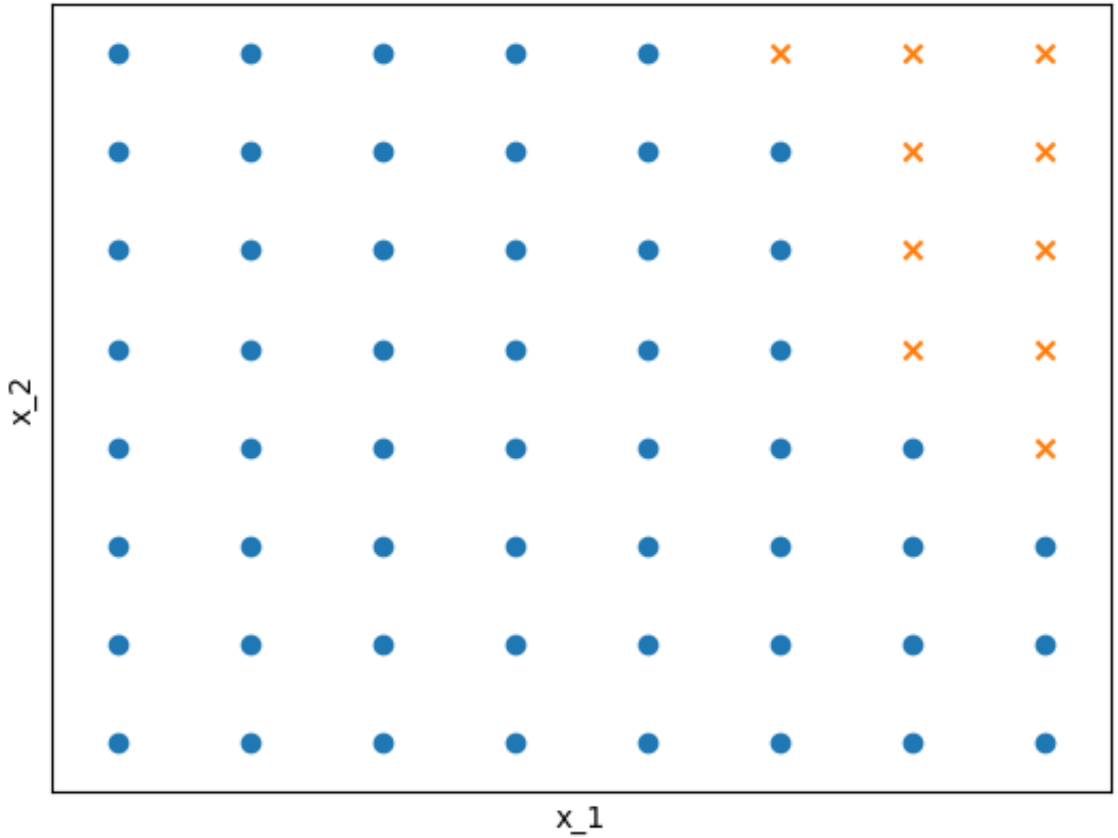
2.2 Consider the set of points $\{(x_1, x_2) : x_1, x_2 \in \{-4, -3, -2, -1, 0, 1, 2, 3, 4\}\}$ and classify each of them according to the naive Bayesian rule.

```
# Activity 2.2 (to be completed)

# define grid of points to be classified
x1, x2 = np.meshgrid(np.arange(-4,4), np.arange(-4, 4))
X_plot = np.c_[x1.cc(), x2.ravel()]

sigmax1_class1 = sigma_class1[0,0]
sigmax2_class1 = sigma_class1[1,1]

sigmax1_class2 = sigma_class2[0,0]
sigmax2_class2 = sigma_class2[1,1]

# Naive Bayesian classifier
prediction_nb = []
for i in range(X_plot.shape[0]):
  # to be completed
  #prob_1 = ...
  #prob_2 = ...
  #if ... :
  #  prediction_nb.append(1)
  #else :
  #  prediction_nb.append(2)

prediction_nb = np.array(prediction_nb)

# plot results
plt.figure()
```

```
plt.scatter(X_plot[np.where(prediction_nb==1),0], X_plot[np.where(prediction_nb==1),1], marker='o')
plt.scatter(X_plot[np.where(prediction_nb==2),0], X_plot[np.where(prediction_nb==2),1], marker='x')
plt.xlabel('x_1')
plt.ylabel('x_2')
plt.xlim(x1.min()-0.5, x1.max()+0.5)
plt.ylim(x2.min()-0.5, x2.max()+0.5)
plt.xticks(())
plt.yticks(())
plt.show()
```

```
# Activity 2.2 (solution)

# define grid of points to be classified
x1, x2 = np.meshgrid(np.arange(-4,4), np.arange(-4, 4))
X_plot = np.c_[x1.ravel(), x2.ravel()]

sigmax1_class1 = sigma_class1[0,0]
sigmax2_class1 = sigma_class1[1,1]

sigmax1_class2 = sigma_class2[0,0]
sigmax2_class2 = sigma_class2[1,1]

# Naive Bayesian classifier
prediction_nb = []
for i in range(X_plot.shape[0]):
  prob_1 = P_class1 * gaussian1d(X_plot[i,0],mu_class1[0],sigmax1_class1) * gaussian1d(X_plot[i,1],mu_class1[1],sigma
  prob_2 = P_class2 * gaussian1d(X_plot[i,0],mu_class2[0],sigmax1_class2) * gaussian1d(X_plot[i,1],mu_class2[1],sigma
  if prob_1 > prob_2 :
    prediction_nb.append(1)
  else :
    prediction_nb.append(2)
```

```
prediction_nb = np.array(prediction_nb)

# plot results
plt.figure()
plt.scatter(X_plot[np.where(prediction_nb==1),0], X_plot[np.where(prediction_nb==1),1], marker='o')
plt.scatter(X_plot[np.where(prediction_nb==2),0], X_plot[np.where(prediction_nb==2),1], marker='x')
plt.xlabel('x_1')
plt.ylabel('x_2')
plt.xlim(x1.min()-0.5, x1.max()+0.5)
plt.ylim(x2.min()-0.5, x2.max()+0.5)
plt.xticks(())
plt.yticks(())
plt.show()
```

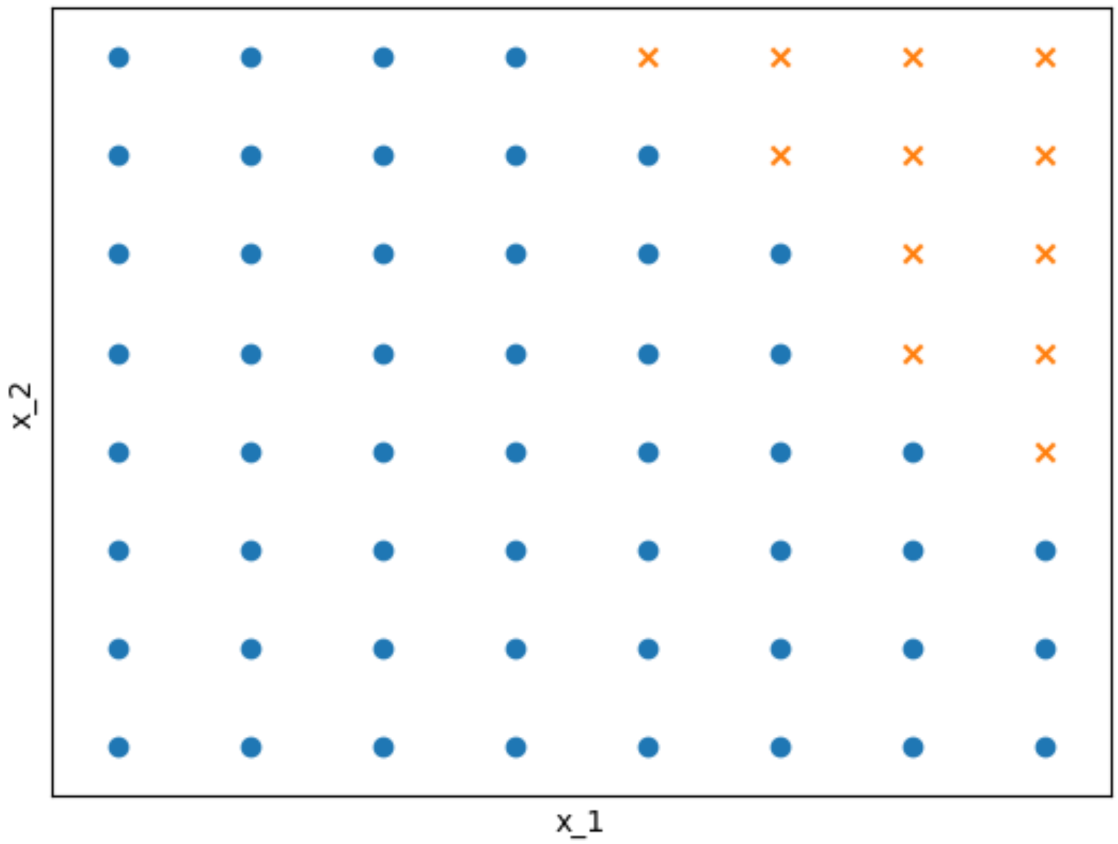


2.3 Repeat the classification process for the above data, but now using the Bayesian classifier rule.

```
# Activity 2.3 (to be completed)

# Bayesian classifier
prediction = []
for i in range(X_plot.shape[0]):
  # to be completed
  #prob_1 = ...
  #prob_2 = ...
  #if ... :
  #  prediction.append(1)
  #else :
  #  prediction.append(2)

prediction = np.array(prediction)

plt.figure()
plt.scatter(X_plot[np.where(prediction==1),0], X_plot[np.where(prediction==1),1], marker='o')
plt.scatter(X_plot[np.where(prediction==2),0], X_plot[np.where(prediction==2),1], marker='x')
plt.xlabel('x_1')
plt.ylabel('x_2')
plt.xlim(x1.min()-0.5, x1.max()+0.5)
plt.ylim(x2.min()-0.5, x2.max()+0.5)
plt.xticks(())
plt.yticks(())
plt.show()
```

```
# Activity 2.3 (solution)

# Bayesian classifier
prediction = []
```

```
for i in range(X_plot.shape[0]):
  prob_1 = P_class1 * gaussian(X_plot[i,0:2],mu_class1,sigma_class1)
  prob_2 = P_class2 * gaussian(X_plot[i,0:2],mu_class2,sigma_class2)
  if prob_1 > prob_2 :
    prediction.append(1)
  else :
    prediction.append(2)

prediction = np.array(prediction)

plt.figure()
plt.scatter(X_plot[np.where(prediction==1),0], X_plot[np.where(prediction==1),1], marker='o')
plt.scatter(X_plot[np.where(prediction==2),0], X_plot[np.where(prediction==2),1], marker='x')
plt.xlabel('x_1')
plt.ylabel('x_2')
plt.xlim(x1.min()-0.5, x1.max()+0.5)
plt.ylim(x2.min()-0.5, x2.max()+0.5)
plt.xticks(())
plt.yticks(())
plt.show()
```


x_2
x_1