Machine Learning

Summer Semester 2019, Homework 2

Prof. Dr. J. Peters, H. Abdulsamad, S. Stark, D. Koert

TECHNISCHE UNIVERSITÄT DARMSTADT

Total points: 90 + 15 bonus
Due date: Friday, 14 June 2019 17:00
Hand in a PDF over Moodle **and** a printed version to the postbox at (S2/02 | E315)

Name, Surname, ID Number **Steffen Schäfer, 2635897** **Peter Nickl, 1941346**

Problem 2.1 Bayesian Decision Theory [20 Points]

In this exercise, we consider data generated by a mixture of two Gaussian distributions with parameters $\{\mu_1, \sigma_1\}$ and $\{\mu_2, \sigma_2\}$. Each Gaussian represents a class labeled C_1 and C_2, respectively.

a) **Optimal Boundary [4 Points]**

Explain in one short sentence what Bayesian Decision Theory is. What is its goal? What condition does hold at the optimal decision boundary? When do we decide for class C_1 over C_2?

In Bayesian decision theory a decsion is made using the degree of belief in an outcome.

It's goal is it to minimze the risk presented in a loss funciton.

At the decision boundary $p(C_1|x) = p(C_2|x)$ holds true.

We decide for C_1 over C_2 if $P(C_1|x) > p(C_2|y)$

b) **Decision Boundaries [8 Points]**

If both classes have equal prior probabilities $p(C_1) = p(C_2)$ and the same variance $\sigma_1 = \sigma_2$, derive the decision boundary x^* analytically as a function of the two means μ_1 and μ_2.

Since the prior are equal following equations holds true we decide for C_1 if:

$$\frac{p(x|C_1)}{p(x|C_2)} > 1 \tag{8}$$

At the point of the decision boundary we can say:

$$p(x|C_1) = p(x|C_2) \tag{9}$$

Both datasets are gausian distributed.

$$\mathcal{N}\left(x|\mu_1,\sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_1)^2\right) \tag{10}$$

$$\mathcal{N}\left(x|\mu_2,\sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_2)^2\right) \tag{11}$$

Using equation 10 and 11 in equation 9 we get:

$$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_1)^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_2)^2\right) \tag{12}$$

$$(x-\mu_1)^2 = (x-\mu_2)^2 \tag{13}$$

$$x = \frac{\mu_1+\mu_2}{2} \tag{14}$$

c) **Different Misclassification Costs [8 Points]**

Assume $\mu_1 > 0$, $\mu_1 = 2\mu_2$, $\sigma_1 = \sigma_2$ and $p(C_1) = p(C_2)$. If misclassifying sample $x \in C_2$ as class C_1 is three times more expensive than the opposite, how does the decision boundary change? Derive the boundary analytically. (There is no cost for correctly classifying samples.)

We want to minimize following risk:

$$\frac{p(x|C_1)}{p(x|C_2)} > \frac{\lambda_{12} - \lambda_{22}}{\lambda_{21} - \lambda_{11}} \cdot \frac{p(C_2)}{p(C_1)} \tag{26}$$

With

$$\lambda_{ij} = \lambda\left(\alpha_i|\alpha_j\right) \tag{27}$$

The main diagonal of λ so λ_{ii} is the cost for correctly classifying. In our case this is zero, so $\lambda_{11} = \lambda_{22} = 0$. Now its stated that missclassifying sample $x \in C_2$ as class C_1 is three times more expensive than the opposite, meaning that $\lambda_{12} = 3 \cdot \lambda_{21}$. We can now use this information for equation 27.

$$\frac{p(x|C_1)}{p(x|C_2)} > \frac{3\lambda_{21}}{\lambda_{21}} \cdot \frac{p(C_2)}{p(C_1)} = 3 \tag{28}$$

Now we can use the equations for the gausian distibution 10 and 11 and get:

$$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_1)^2\right) = 3 \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu_2)^2\right) \tag{29}$$

$$\exp\left(-\frac{1}{2\sigma^2}(x-\mu_1)^2\right) = 3 \cdot \exp\left(-\frac{1}{2\sigma^2}(x-\mu_2)^2\right) \tag{30}$$

Using the log to get rid of the exponential function.

$$-\frac{1}{2\sigma^2}(x-\mu_1)^2 = \ln(3) - \frac{1}{2\sigma^2}(x-\mu_2)^2 \tag{31}$$

We now use $\mu_1 = 2\mu_2$ and continue.

$$-\frac{1}{2\sigma^2}\left(x^2 - 4x\mu_2 + 4\mu_2^2\right) + \frac{1}{2\sigma^2}\left(x^2 - 2x\mu_2 + \mu_2^2\right) = ln(3) \tag{32}$$

$$-\frac{1}{2\sigma^2}\left(x^2 - 4x\mu_2 + 4\mu_2^2\right) + \frac{1}{2\sigma^2}\left(-x^2 + 2x\mu_2 - \mu_2^2\right) = ln(3) \tag{33}$$

Transforming the terms inside the brackets brings us to

$$-2x\mu_2 + 3\mu_2^2 = ln(3) \cdot \left(-2\sigma^2\right) \tag{34}$$

$$-2x\mu_2 = -2ln(3)\sigma^2 - 3\mu_2^2 \tag{35}$$

After transposing this equation we get:

$$x = \frac{3\mu_2^2 + 2\ln(3) \cdot \sigma^2}{2\mu_2} \tag{36}$$

Problem 2.2 Density Estimation [30 Points + 15 Bonus]

In this exercise, you will use the datasets densEst1.txt and densEst2.txt. The datasets contain 2D data belonging to two classes, C_1 and C_2.

a) **Gaussian Maximum Likelihood Estimation [10 Points]**

Derive the ML estimate for the mean and covariance of the **multivariate** Gaussian distribution. Start your derivations with the function you optimize. Assume that you can collect i.i.d data. (Hint: you can find many matrix identities on the Matrix Cookbook (https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf) and at http://en.wikipedia.org/wiki/Matrix_calculus.)

The Multivariate Gaussian Distribution is defined as

$$f_x(\mu|\Sigma) = \frac{1}{\sqrt{(2\pi)^p \, det(\Sigma)}} exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right) \tag{59}$$

Σ is the Covariance Matrix defined as:

$$\begin{pmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 \\ \sigma_{xy}^2 & \sigma_{yy}^2 \end{pmatrix} \tag{60}$$

The resulting likelihood function of $f_x(x)$ is:

$$L(\mu,\Sigma) = \prod_{n=1}^{N} f_x(\mu|\Sigma) \tag{61}$$

We now take the log of the likelihood function.

$$\log L(\mu,\Sigma) = \sum_{n=1}^{N} \log f_x(\mu|\Sigma) \tag{62}$$

Resulting in the log likelihood function for a multivariate gausian distribution.

$$\log L(\mu,\Sigma) = \sum_{n=1}^{N} \log\left(\frac{1}{\sqrt{(2\pi)^p \, det(\Sigma)}} exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)\right) \tag{63}$$

Now we want to derive the mean and covariance for this distribution. Starting with the mean, we need to maximize the partial equation.

$$\frac{\partial L}{\partial \mu} = \underbrace{\sum_{n=1}^{N} \log\left(\frac{1}{\sqrt{(2\pi)^p \, det(\Sigma)}}\right)'}_{0} + \sum_{n=1}^{N}\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)' \tag{64}$$

For matrix Calculus we use following rules:

$$\frac{\partial (Xa+b)^T C (Xa+b)}{\partial x} = \left(C + C^T\right)(Xa+b)\,a^T \tag{65}$$

With $X = x$, $a = 1$, thus $a^T = 1$ and $b = -\mu$. We get

$$\sum_{n=1}^{N}\left(\Sigma + \left(\Sigma^{-1}\right)^T (x-\mu)\right) = 0 \tag{66}$$

Since Σ is a symmetrical Matrix $\Sigma + \Sigma^T = 2 \cdot \Sigma$ holds true. This matrix is also constant, meaning its independend on the sum and can be written before it.

$$2\Sigma^{-1} \cdot \sum_{n=1}^{N} (x - \mu) = 0 \tag{67}$$

$$\sum_{n=1}^{N} x_i = \sum_{n=1}^{N} \mu \tag{68}$$

Since μ is a constant we can write the sum over a constant as $N \cdot \mu$

$$\sum_{n=1}^{N} x_i = N \cdot \mu \tag{69}$$

Resulting in the following equation for the mean:

$$\mu = \frac{1}{N} \sum_{n=1}^{N} x_i \tag{70}$$

Now we want to derive the maximum Likelihood estimate for the covariance of the multivariate Gaussian distribution. Again we Start with Equation (63). For this derivation the will the following rules for matrix calculus.

$$\frac{\partial}{\partial A} \log(\det(A)) = \left(A^{-1}\right)^T \tag{71}$$

$$\frac{\partial}{\partial A} x^T A x = x x^T \tag{72}$$

With $A = \Sigma$ and $x = (x - \mu)$. First we split the log of gaussian equation up into three part for easier derivation.

$$\frac{\partial L}{\partial \Sigma} = \underbrace{\log\left(\frac{1}{\sqrt{2\pi}^P}\right)'}_{a} + \sum_{n=1}^{N} \left(\underbrace{\log\left(\frac{1}{\sqrt{\det(\Sigma)}}\right)'}_{b} + \underbrace{\log\exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)'}_{c} \right) \tag{73}$$

For a:

$$\frac{\partial L_a}{\partial \Sigma} = 0 \tag{74}$$

For b:

$$\log\left(\frac{1}{\sqrt{\det(\Sigma)}}\right) = \log\left(\frac{1}{\det(\Sigma)^{\frac{1}{2}}}\right) = \frac{1}{2} \cdot \log\left(\frac{1}{\det(\Sigma)}\right) \tag{75}$$

$$\frac{\partial L_b}{\partial \Sigma} = \frac{1}{2} \cdot \frac{1}{(\Sigma^{-1})^T} = \frac{1}{2} \cdot \Sigma \tag{76}$$

For c:

$$\frac{\partial L_c}{\partial \Sigma} = -\frac{1}{2} (x - \mu) \cdot (x - \mu)^T \tag{77}$$

In total we get the following derivative:

$$\frac{\partial L}{\partial \Sigma} = \sum_{n=1}^{N} \frac{1}{2} \Sigma - \sum_{n=1}^{N} \left(\frac{1}{2} (x-\mu) \cdot (x-\mu)^T \right) = 0 \tag{78}$$

Multiplied by $\frac{1}{2}$:

$$\sum_{n=1}^{N} \Sigma = \sum_{n=1}^{N} \left((x-\mu) \cdot (x-\mu)^T \right) \tag{79}$$

Since Σ is constant we can write $\sum_{n=1}^{N} \Sigma = N \cdot \Sigma$. Resulting in the final equation for the covariance

$$\Sigma = \frac{1}{N} \sum_{n=1}^{N} (x_i - \mu) \cdot (x_i - \mu)^T \tag{80}$$

b) **Prior Probabilities [2 Points]**

Compute the prior probability of each class from the dataset.

We count for class 1: $n_1 = 239$ and for class 2: $n_2 = 761$. Resulting in following priors

$$p(C_1) = \frac{239}{239 + 761} = 0.239 \tag{83}$$

$$p(C_2) = \frac{761}{239 + 761} = 0.761 \tag{84}$$

c) **Biased ML Estimate [5 Points]**

Define the bias of an estimator and write how we can compute it. Then calculate the biased and unbiased estimates of the conditional distribution $p(x|C_i)$, assuming that each class can be modeled with a Gaussian distribution. Which parameters have to be calculated? Show the final result and attach a snippet of your code. Do not use existing functions, but rather implement the computations by yourself!

The bias of an estimator Θ measures the difference between the expected value of an estimator and its' true value. It is is defined as:

$$Bias\left(\widehat{\theta}\right) = E\left(\widehat{\theta}\right) - \theta$$

Assuming a Gaussian distribution, the biased and unbiased estimates of the conditional distributions can be calculated using the following formula:

$$\bar{\mathbf{x}}_{unbiased} = \frac{1}{N}\sum_{i=1}^{n}\mathbf{x}_i$$

$$\widehat{\boldsymbol{\Sigma}}_{unbiased} = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^{\mathrm{T}}$$

$$\widehat{\boldsymbol{\Sigma}}_{biased} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^{\mathrm{T}}$$

We thus obtain the following values for the given dataset.

$$\bar{\mathbf{x}}_{1,unbiased} = \begin{bmatrix} -0.70681374 \\ -0.81343083 \end{bmatrix}$$

$$\widehat{\boldsymbol{\Sigma}}_{1,unbiased} = \begin{bmatrix} 9.05742302 & 2.6841014 \\ 2.6841014 & 3.61145033 \end{bmatrix}$$

$$\widehat{\boldsymbol{\Sigma}}_{1,biased} = \begin{bmatrix} 9.01952586 & 2.67287085 \\ 2.67287085 & 3.59633965 \end{bmatrix}$$

$$\bar{\mathbf{x}}_{2,unbiased} = \begin{bmatrix} 3.98534252 \\ 3.98438364 \end{bmatrix}$$

$$\widehat{\boldsymbol{\Sigma}}_{2,unbiased} = \begin{bmatrix} 4.18087542 & 0.02761954 \\ 0.02761954 & 2.75658555 \end{bmatrix}$$

$$\widehat{\boldsymbol{\Sigma}}_{2,biased} = \begin{bmatrix} 4.1753815 & 0.02758324 \\ 0.02758324 & 2.75296323 \end{bmatrix}$$

The code for calculating the estimators is attached.

```python
# import necessary Python packages
import os
import numpy as np

# set the working directory
os.chdir("C:/Users/pistl/Desktop/hw2/")

# print the current working directory
os.getcwd()

# load data
densEst1 = np.loadtxt(fname = "C:/Users/pistl/Desktop/hw2/dataSets/densEst1.cvs")
densEst2 = np.loadtxt(fname = "C:/Users/pistl/Desktop/hw2/dataSets/densEst2.cvs")
```

```
def sample_mean(data):
    N = np.size(data,0)
    column_sums = np.sum(data, axis=0)
    means = 1/N * column_sums
    return means

def sample_variance_biased(data,means):
    N = np.size(data,0)
    x = data - means
    variances_biased = 1/N * np.matmul(np.transpose(x),x)
    return variances_biased

def sample_variance_unbiased(data,means):
    N = np.size(data,0)
    denominator = N - 1
    x = data - means
    variances_biased = 1/denominator * np.matmul(np.transpose(x),x)
    return variances_biased

print('ESTIMATORS OF DATASET densEst1\n')
means = sample_mean(densEst1)
print('means\n',means)
print('\nvariance_biased\n',sample_variance_biased(densEst1,means))
print('\nvariance_unbiased\n',sample_variance_unbiased(densEst1,means))

print('--------------------------------------')

print('ESTIMATORS OF DATASET densEst2\n')
means = sample_mean(densEst2)
print('means\n',means)
print('\nvariance_biased\n',sample_variance_biased(densEst2,means))
print('\nvariance_unbiased\n',sample_variance_unbiased(densEst2,means))
```

d) **Class Density [5 Points]**

Using the unbiased estimates from the previous question, fit a Gaussian distribution to the data of each class. Generate a single plot showing the data points and the probability densities of each class. (Hint: use the contour function for plotting the Gaussians.)

Using the unbiased estimates from before, we can plot the class densities and the data points. See Figure 1 for the plot. According to the datasets, the red dots belong to class 1 and the blue dots belong to class 2.

e) **Posterior [8 Points]**

In a single graph, plot the posterior distribution of each class $p(C_i|x)$ and show the decision boundary.

The figures 2 and 3 show the posterior distribution of both classes in a single plot. Figure 2 shows a contour plot, whereas figure 3 shows a 3d view.
The figures 4 and 5 display the decision boundary, with two different scales for the x- and y-axes. The colour yellow indicates a decision for class 1 and purple encodes a decision for class 2.

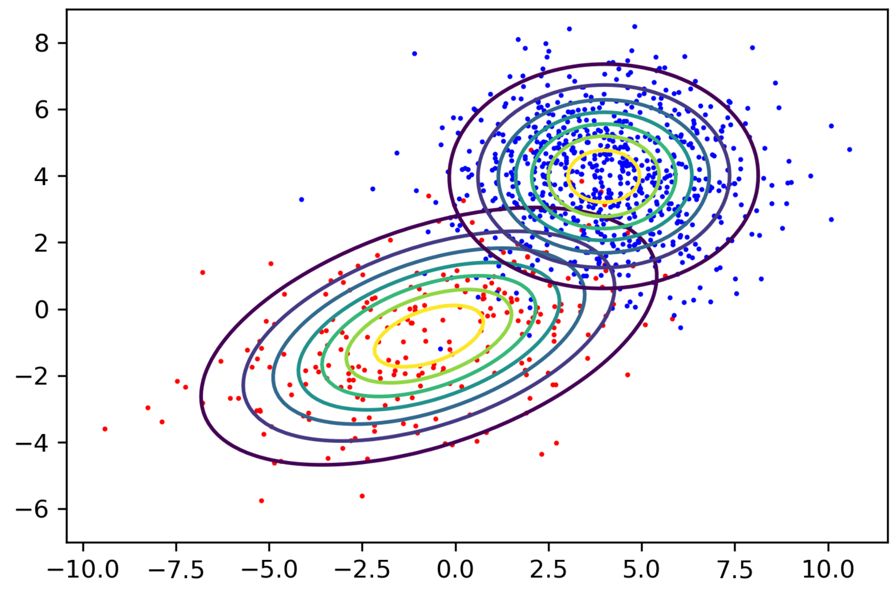


Figure 1: Plot of the class densities and data points.

f) **Bayesian Estimation [15 Bonus Points]**

State the generic case of Bayesian linear regression with data $< \boldsymbol{X}, \boldsymbol{Y} >$ and parameters $\boldsymbol{\theta}$. What do we assume about the data, the model and the parameters?
Formulate the posterior distribution for your model parameters given the data, i.e., $p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{Y})$, and derive its mean and covariance, assuming that the model of the output variable is a Gaussian distribution with a fixed variance.
What do we do when we want to predict a new point?
Which are the advantages of being Bayesian?

For this task we follow the these technical notes: Thomas P. Minka (2010) - Bayesian linear regression.
The data is is modeled in the following way: An input vector $\mathbf{x}$ *of length m multiplies a coefficient matrix* $\boldsymbol{\theta}$ *in order to produce and output vector* $\mathbf{y}$ *of length d, with Gaussian noise* $\mathbf{e}$ *added. The noise* $\mathbf{e}$ *has zero mean and a variance* $\mathbf{V}$.

$$\mathbf{y} = \boldsymbol{\theta}\mathbf{x} + \mathbf{e}$$
$$\mathbf{e} \sim \mathcal{N}(0, \mathbf{V})$$

This is equivalent to the following conditional distribution for $\mathbf{y}$*:*

$$p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta},\mathbf{V}) \sim \mathcal{N}(\boldsymbol{\theta}\mathbf{x},\mathbf{V})$$

The set of pairs $D = \{(\mathbf{y}_1,\mathbf{x}_1),\ldots,(\mathbf{y}_N,\mathbf{x}_N)\}$ *are independent and identically distributed, which leads to:*

$$\begin{aligned} p(\mathbf{Y}|\mathbf{X},\boldsymbol{\theta},\mathbf{V}) &= \prod_i p(\mathbf{y}_i|\mathbf{x}_i,\boldsymbol{\theta},\mathbf{V}) \\ &= \frac{1}{|2\pi\mathbf{V}|^{N/2}} \exp\left(-\frac{1}{2}\operatorname{tr}\left(\mathbf{V}^{-1}[\boldsymbol{\theta}\mathbf{X}\mathbf{X}^{\mathrm{T}}\boldsymbol{\theta}^{\mathrm{T}} - 2\mathbf{Y}\mathbf{X}^{\mathrm{T}}\boldsymbol{\theta}^{\mathrm{T}} + \mathbf{Y}\mathbf{Y}^{\mathrm{T}}]\right)\right) \end{aligned} \tag{87}$$

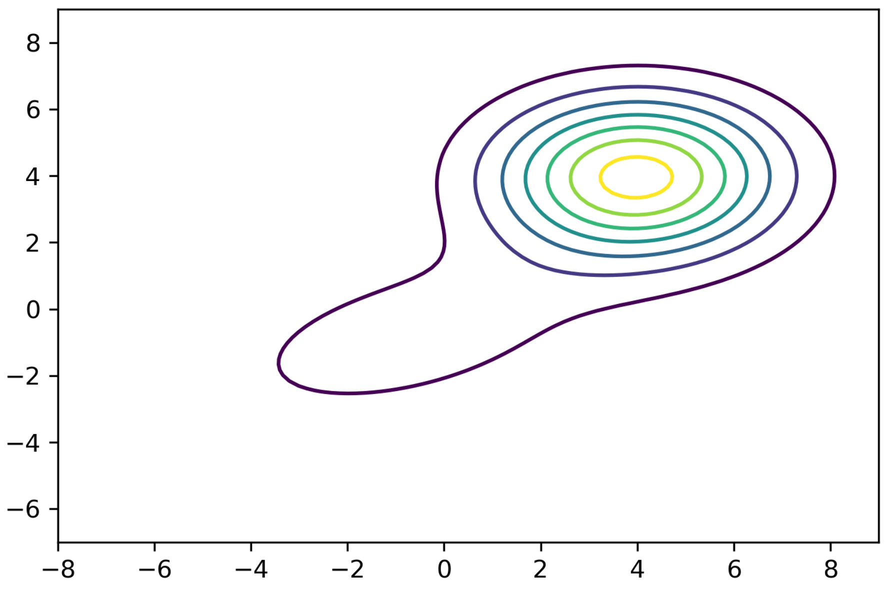


Figure 2: Contour plot of the posterior distribution of each class in a single graph.

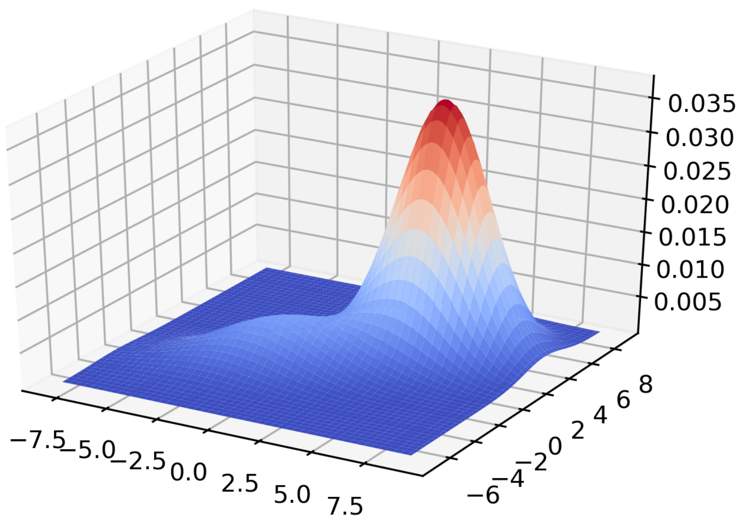


Figure 3: 3d plot of the posterior distribution of each class in a single graph.

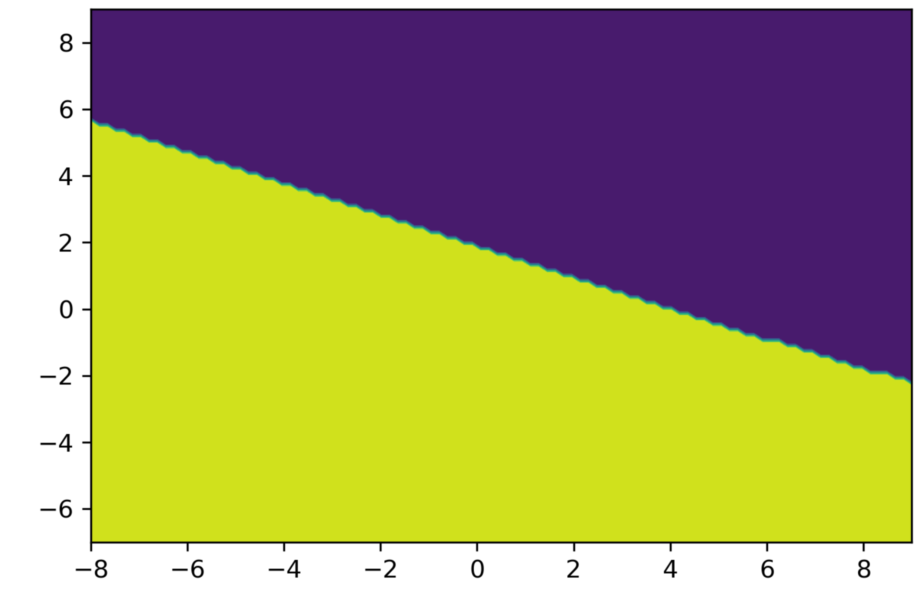


Figure 4: Decision boundary with small scale of the x- and y-axes.

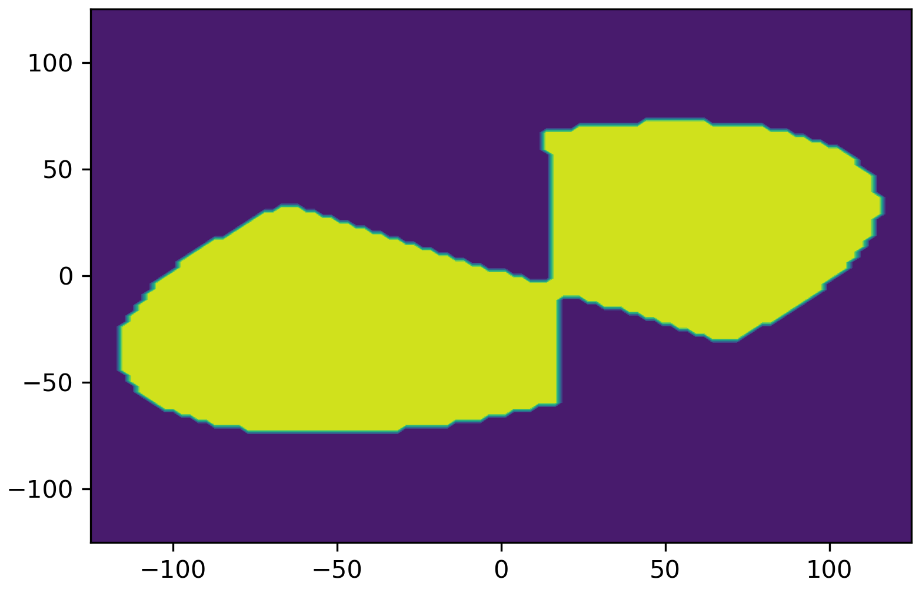


Figure 5: Decision boundary with large scale of the x- and y-axes

A conjugate prior for the coeffiecients $\boldsymbol{\theta}$ is the matrix-normal density with parameters $\mathbf{M}(dxm)$, $\mathbf{V}(dxd)$ and $\mathbf{K}(mxm)$:

$$\begin{aligned} p(\boldsymbol{\theta}) &\sim \mathcal{N}(\mathbf{M},\mathbf{V},\mathbf{K}) \\ &= \frac{|\mathbf{K}|^{d/2}}{|2\pi\mathbf{V}|^{m/2}} \exp\left(-\frac{1}{2}\operatorname{tr}\left((\boldsymbol{\theta}-\mathbf{M})^{\mathrm{T}}\mathbf{V}^{-1}(\boldsymbol{\theta}-\mathbf{M})\mathbf{K}\right)\right) \end{aligned} \tag{88}$$

If we use the following abbreviations:

$$\mathbf{S}_{xx} = \mathbf{X}\mathbf{X}^{\mathrm{T}} + \mathbf{K}$$

$$\mathbf{S}_{yx} = \mathbf{Y}\mathbf{X}^{\mathrm{T}} + \mathbf{M}\mathbf{K}$$

$$\mathbf{S}_{yy} = \mathbf{Y}\mathbf{Y}^{\mathrm{T}} + \mathbf{M}\mathbf{K}\mathbf{M}^{\mathrm{T}}$$

$$\mathbf{S}_{y|x} = \mathbf{S}_{yy} - \mathbf{S}_{yx}\mathbf{S}_{xx}^{-1}\mathbf{S}_{yx}^{\mathrm{T}}$$

Under the assumption that the output is gaussian with fixed variance, he posterior for $\boldsymbol{\theta}$ then is matrix normal:

$$p(\boldsymbol{\theta}|D,\mathbf{V}) \sim \mathcal{N}\left(\mathbf{S}_{yx}\mathbf{S}_{xx}^{-1},\mathbf{V},\mathbf{S}_{xx}\right)$$

This can be derived by taking the likelihood (87) times the conjugate prior (88):

$$\begin{aligned} p(\mathbf{Y},\boldsymbol{\theta}|\mathbf{X},\mathbf{V}) &\propto \exp\left(-\frac{1}{2}\operatorname{tr}\left(\mathbf{V}^{-1}\left(\boldsymbol{\theta}\mathbf{S}_{xx}\boldsymbol{\theta}^{\mathrm{T}} - 2\mathbf{S}_{yx}\boldsymbol{\theta}^{\mathrm{T}} + \mathbf{S}_{yy}\right)\right)\right) \\ &\propto \exp\left(-\frac{1}{2}\operatorname{tr}\left(\mathbf{V}^{-1}\left(\left(\boldsymbol{\theta}-\mathbf{S}_{yx}\mathbf{S}_{xx}^{-1}\right)\mathbf{S}_{xx}\left(\boldsymbol{\theta}-\mathbf{S}_{yx}\mathbf{S}_{xx}^{-1}\right)^{\mathrm{T}} + \mathbf{S}_{y|x}\right)\right)\right) \end{aligned}$$

The advantages of being Bayesian are, that the pararameters are regarded as random variables and a prior is put on these. As a consequence the uncertainty of parameter values is incorporated in the model. This leads to automatic regularization and model selection according to the data and avoids over-/ or underfitting. There is no need for separate model selection criteria as BIC and AIC. Furthermore bayesian estimation provides a natural way of including prior knowledge with incoming data, in order to obtain a posterior. This posterior can be used as prior for new decisions.

Problem 2.3 Non-parametric Density Estimation [20 Points]

In this exercise, you will use the datasets `nonParamTrain.txt` for training and `nonParamTest.txt` for evaluating the performance of your model.

a) **Histogram [4 Points]**

Compute and plot the histograms using 0.02, 0.5, 2.0 size bins of the training data. Intuitively, indicate which bin size performs the best and explain why. Knowing only these three trials, would you be sure that the one you picked is truly the best one? Attach the plot of the histograms and a snippet of your code.

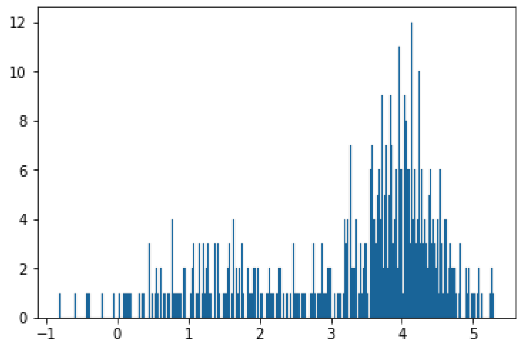
The first figure 6a shows a Histogram of the data with bin size 0.02. The "spicky" behavior and empty bins which are conssistend throughout the Histogram, indicate that the bin size has been chosen to to small (to much Variance).

In the next Figure 6b you can imagine a trend for the underlying distribution. This indicates a good bin size.

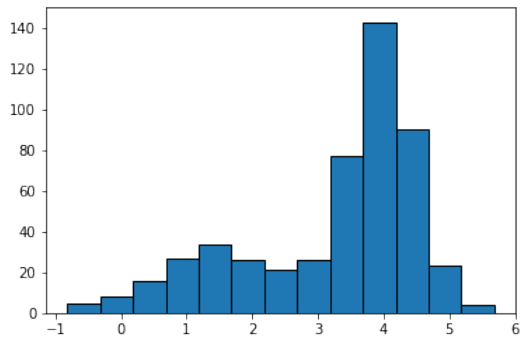
The third histogram 7a also schows a trend of the underlying function, but compared to figure 6b some of details get lost in the representation. Meaning that the bin size has been choosen to big (to much Bias).

Bonus: This last histogram 7b is an histogram with the number of bins determined by Sturges Rule. It's shape is similiar to figure (2), only beeing a bit more smooth. However Sturges rule is critized to smooth the Histogram to much (Hyndman, 1995) and should be only used as a rule of thumb. In this case it adds futher believe into a bin size of about 0.5

However in the end for these four bin sizes, we cant be completly sure that the one choose is the best, but we have atleast good evidence for it.

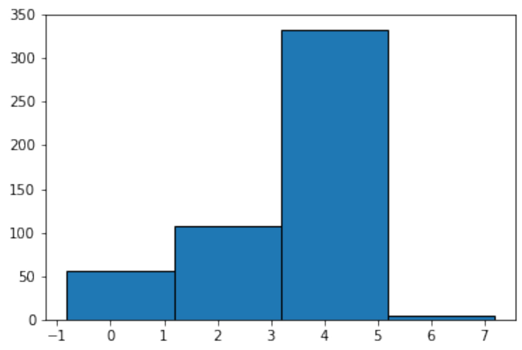


(a) Bin Size 0.02.

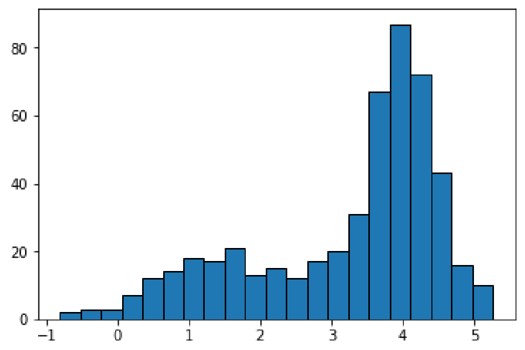


(b) Bin Size 0.5.

Figure 6: First two Histograms



(a) Bin Size 2



(b) Bin size calculated with Sturges Rule.

Figure 7: Last two Histograms

Snippet of the code used:

```
import numpy as np
train = np.loadtxt("nonParamTrain.txt");

import matplotlib.pyplot as plt
binwidth = 0.02
bin_sequence = np.arange(min(train), max(train) + binwidth, binwidth)
plt.hist(train, bins=bin_sequence, edgecolor="black" ,linewidth=0.1);
```

b) **Kernel Density Estimate [6 Points]**

Compute the probability density estimate using a Gaussian kernel with $\sigma = 0.03$, $\sigma = 0.2$ and $\sigma = 0.8$ of the training data. Compute the log-likelihood of the data for each case, compare them and show which parameter performs the best. Generate a single plot with the different density estimates and attach it to your solutions. Plot the densities in the interval $x \in [-4, 8]$, attach a snippet of your code and comment the results.

The log-likelihood for the first Gaussian kernel with $\sigma = 0.03$ *is:* -674.7.

The log-likelihood for the second Gaussian kernel with $\sigma = 0.2$ *is:* -717.0.

The log-likelihood for the third Gaussian kernel with $\sigma = 0.08$ *is:* -795.7.

From campring the log-likelihoods the first kernel should be the best one. But by futher observing the graphs it seems that the second kernel is the best suited for the task.

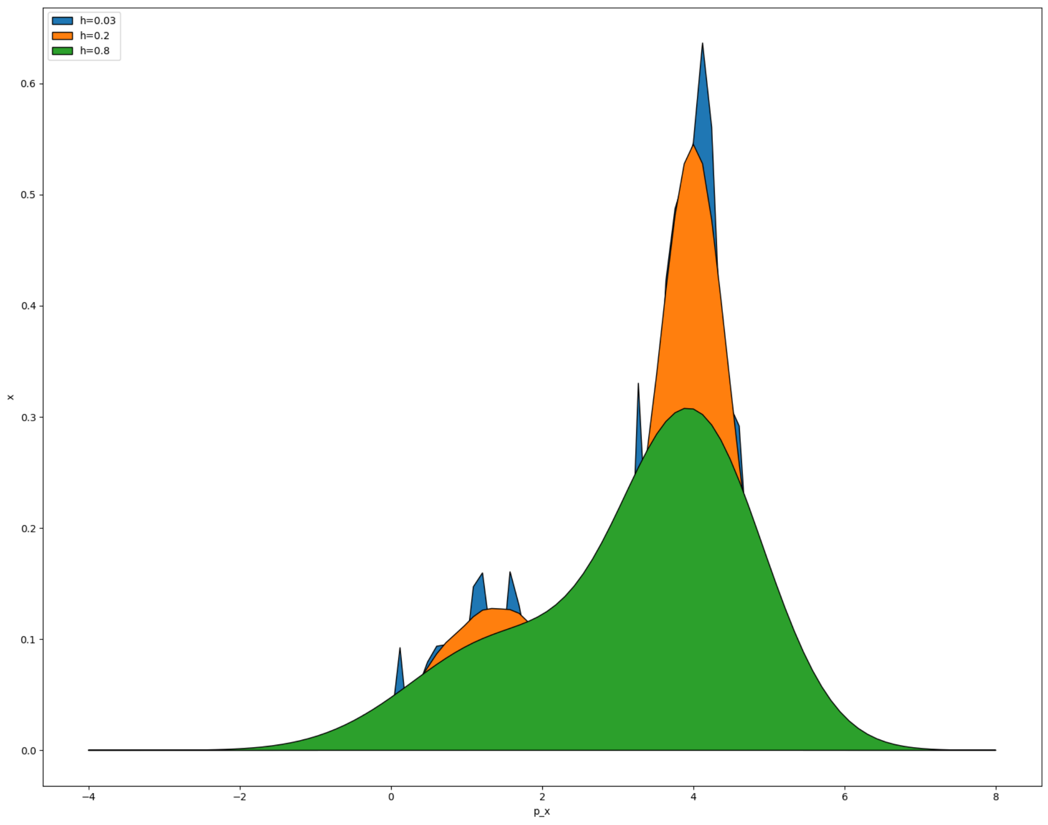
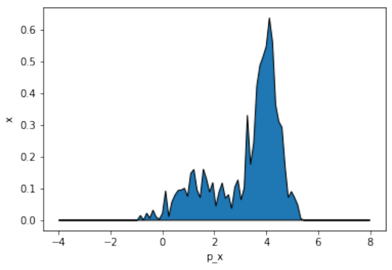
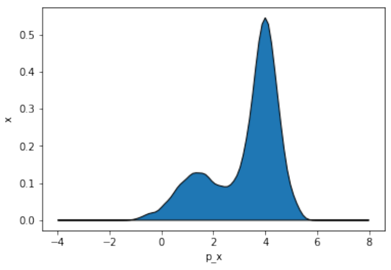


Figure 8: Kernel density estimation for $\sigma = h = [0.03, 0.2, 0.8]$


(a) Kernel bandwidth of 0.03


(b) Kernel bandwidth of 0.2.

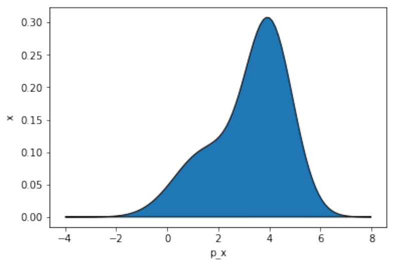

(c) Kernel bandwidth of 0.8.

Figure 9: All the plots seperated

Snippet of the code used:

```
def gaussian(x,h):
    return np.exp(-x**2/(2*h**2))/(h*np.sqrt(2*np.pi))

N = train.size
X_plot = np.linspace(-4, 8, 100)
sum1 = np.zeros(len(X_plot))
sum2 = np.zeros(len(X_plot))
sum3 = np.zeros(len(X_plot))

for i in range(0, N):
    sum1 += ((gaussian(X_plot - train[i], h=0.03)) / N)

# Plot the result
plt.fill(X_plot, sum1, edgecolor="black" ,linewidth=1)
plt.xlabel("p_x")
plt.ylabel("x")

# Calculate the likelihood
likelihood_sum = 0
for i in range(0, N):
    likelihood = 0
    for j in range(0, N):
        likelihood += ((gaussian(train[i] -train[j], h=0.03)) / N)
    likelihood = math.log(likelihood)
    likelihood_sum += likelihood
print(likelihood_sum)
```

c) **K-Nearest Neighbors [6 Points]**

Estimate the probability density with the K-nearest neighbors method with $K = 2, K = 8, K = 35$. Generate a single plot with the different density estimates and attach it to your solutions. Plot the densities in the interval $x \in [-4, 8]$, attach a snippet of your code and comment the results.

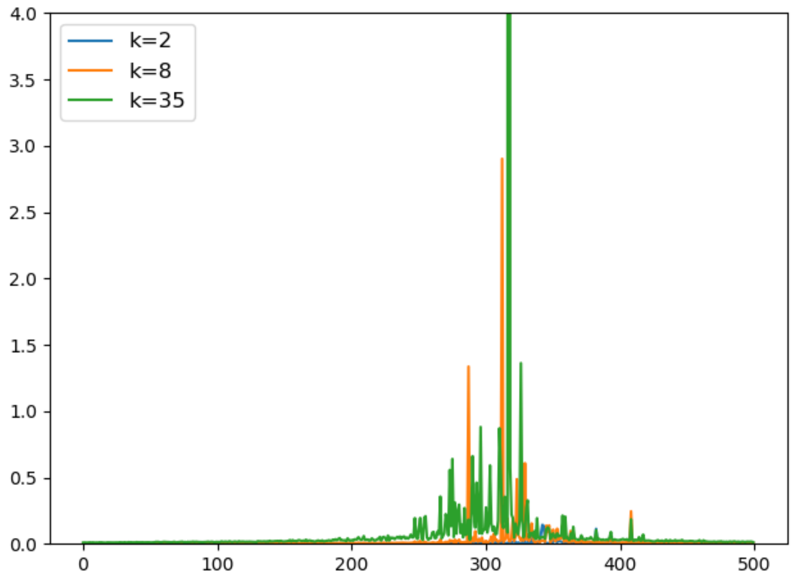


Figure 10: K-Nearest Neighbors for $k = 2, k = 8, k = 35$

```
import numpy as np
import matplotlib.pyplot as plt
import math

train = np.loadtxt("nonParamTrain.txt");
k = 35
N = train.size
distances = np.zeros([N, N - 1])
pd = np.zeros(N)
x_axis = np.linspace(-4, 8, N)

# plot density
for i in range(0, N):
    for ii in range(0, N - 1, ):
        distances[i][ii] = abs(train[i] - train[ii])

    distances[i, :] = np.sort(distances[i, :]);
    pd[i] = distances[i][-k] # k nearest neighbor

result = k / (N * abs(x_axis - pd))
plt.plot(result)

# Compute log likelihood
distances = np.zeros([N, N - 1])
pd = np.zeros(N)
for i in range(0, N):
    for ii in range(0, N - 1, ):
        distances[i][ii] = abs(train[i] - train[ii])

    distances[i, :] = np.sort(distances[i, :]);
    pd[i] = distances[i][-k] # k nearest neighbor

likelihood = 0
for i in range(0, N):
    likelihood += math.log(k / (N * abs(train[i] - pd[i])))
print(likelihood)
```

d) **Comparison of the Non-Parametric Methods [4 Points]**

Estimate the log-likelihood of the testing data using the KDE estimators and the K-NN estimators. Why do we need to test them on a different data set? Compare the log-likelihoods of the estimators w.r.t. both the training and testing sets in a table. Which estimator would you choose?

The following table lists all results for the log-likelihoods of the estimators w.r.t both the training and testing sets in a table. The First three rows are for the training set with $\sigma = [0.03, 0.2, 0.8]$ *and* $k = [2, 8, 35]$ *and the next three for the test set using the same order oder paramterization.*

Actually from the log-likelihoods it seems that all estimators perform badly. However the Kernel Density Estimate using Gaussian kernel has much better results than the knn-Algorithm so its prefferable.

	KDE	*KNN*
1.	*-675*	*-2675*
2.	*-717*	*-875*
3.	*-795*	*-1273*
4.	*-2812*	*-14631*
5.	*-2877*	*-11115*
6.	*3192*	*-4469*

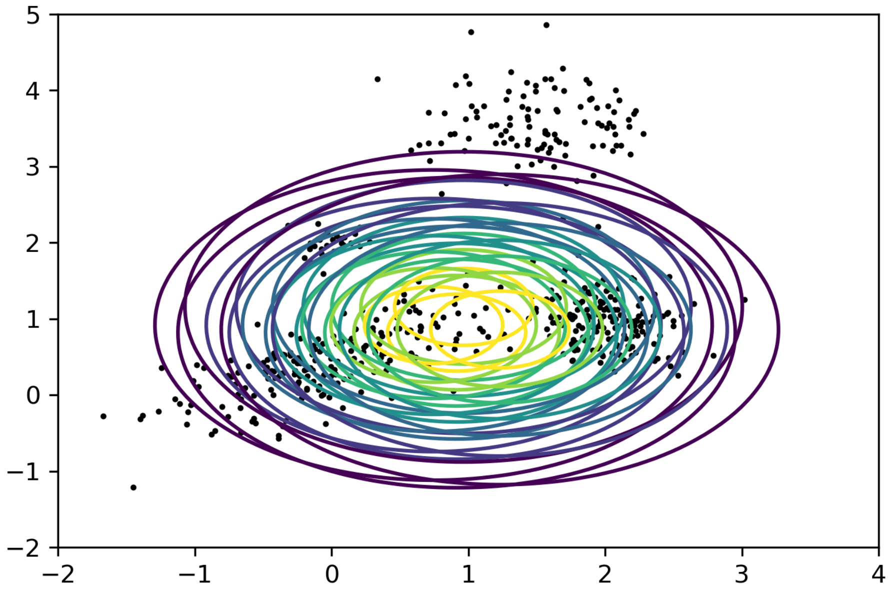


Figure 11: Plot of mixture components and data after 1 iteration.

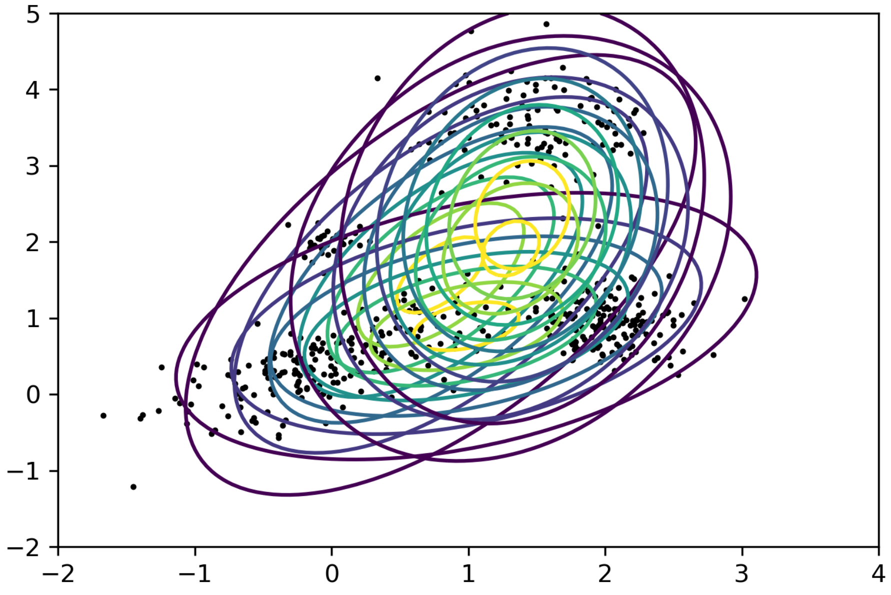


Figure 12: Plot of mixture components and data after 3 iterations.

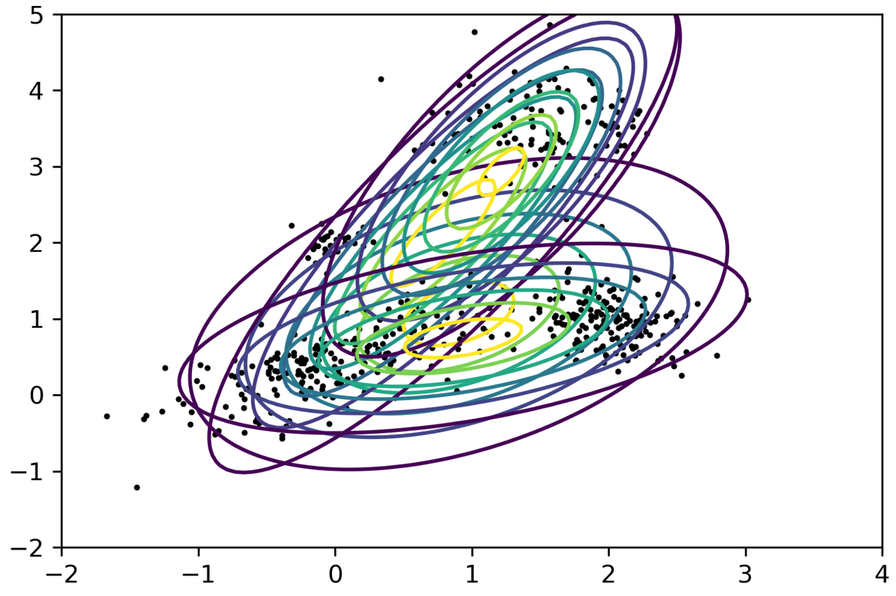


Figure 13: Plot of mixture components and data after 5 iterations.

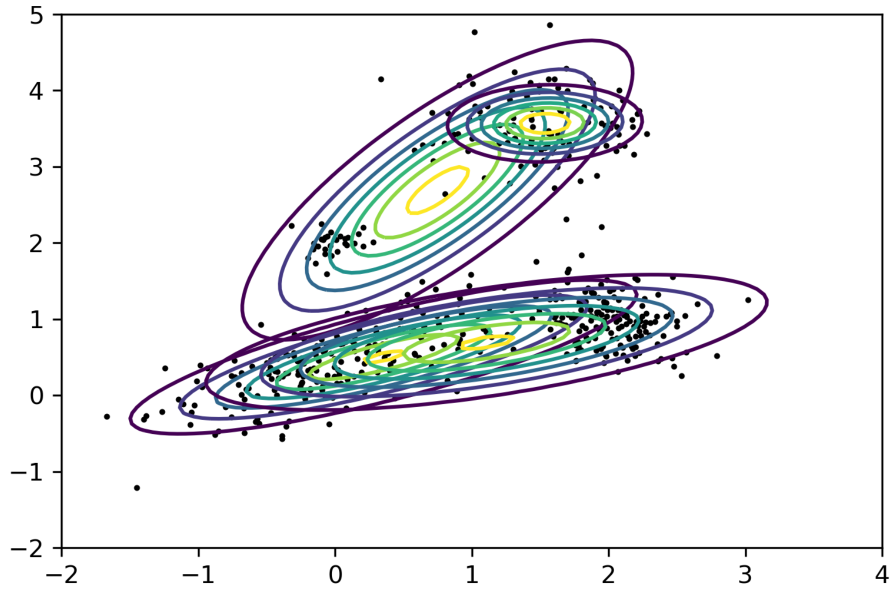


Figure 14: Plot of mixture components and data after 10 iterations.

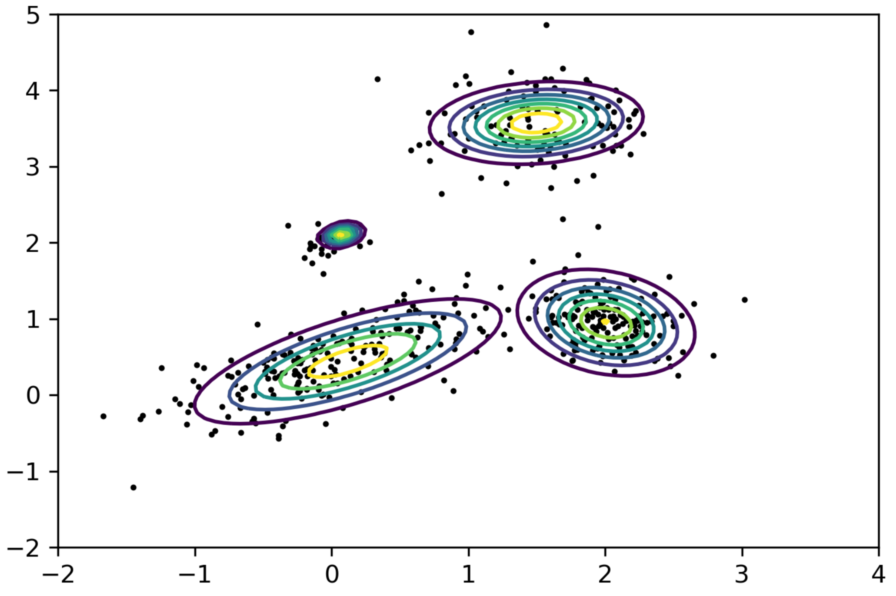


Figure 15: Plot of mixture components and data after 30 iterations.

Problem 2.4 Expectation Maximization [20 Points]

In this exercise, you will use the datasets `gmm.txt`. It contains data from a Gaussian Mixture Model with four 2-dimensional Gaussian distributions.

a) **Gaussian Mixture Update Rules [2 Points]**

Define the model parameters and the update rules for your model. Specify the E- and M-steps of the algorithm.

b) **EM [18 Points]**

Implement the Expectation Maximization algorithm for Gaussian Mixture Models. Initialize your model uniformly. Generate plots at different iterations $t_i \in [1,3,5,10,30]$, showing the data and the mixture components, and plot the log-likelihood for every iteration $t_i = 1:30$. Attach a snippet of your code.

For the iterations $t_i \in [1,3,5,10,30]$*we obtain the following plots with the data and the mixture components. The code of the EM implementation is attached:*

```
import numpy as np
import numpy.random as npr
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
import os

# set the working directory
os.chdir("C:/Users/pistl/Desktop/hw2/")

# print the current working directory
os.getcwd()

# read in data
def sample_GMM():
    data = np.loadtxt(fname = "C:/Users/pistl/Desktop/hw2/dataSets/gmm.txt",usecols=(0, 1), skiprows=1)
    #print(data)
    return data

# Initialize parameters uniformly
def initialize_uniformly(data,D,K):

    mus = npr.uniform(low=0.7, high=1.3, size = [K, D])
    sigmas = np.zeros((K,D,D))
    for k in range(K):
        sigmas[k,:,:] = np.eye(D,D)
    pis = npr.dirichlet(np.ones(K)) #mixture coefficients

    return mus,sigmas,pis


# E-step of EM algorithm
def e_step(data,D,K,mus,sigmas,pis):

    # calculate responsibilities (= probability that data point was generated by cluster k)
    sums = np.zeros(N)
    for n in range(N):
        for k in range(K):
                sums[n] += pis[k] * multivariate_normal.pdf(data[n],mus[k],sigmas[k])
    resp = np.zeros((N,K))
    for n in range(N):
        for k in range(K):
                resp[n,k] = pis[k] * multivariate_normal.pdf(data[n],mus[k],sigmas[k]) / sums[n]
    return resp


# M-step of EM algorithm
def m_step(data,D,K,resp):

    # calculate N_k
    N_k = np.zeros(K)
    N_k = np.sum(resp,axis=0)
    N_sum = np.sum(N_k)

    # update parameters
    # update mu
    mu = np.zeros((K,D))
    for k in range(K):
        for n in range(N):
            mu[k] += resp[n,k] * data[n,:]
        mu[k] = mu[k] / N_k[k]

    # update sigma
    sigma = np.zeros((K,D,D))
    for k in range(K):
        for n in range(N):
            sigma[k,:,:] += resp[n,k] * np.matmul(np.atleast_2d(data[n,:] - mu[k,:]).T,
                 np.atleast_2d(data[n,:] - mu[k,:]))
        sigma[k,:,:] = sigma[k,:,:] / N_k[k]

    # update pi
    pi = np.zeros(K)
    for k in range(K):
        pi[k] = N_k[k] / N_sum

    return mu, sigma, pi


# calculate loglikelihood of the data under the parameter estimation
def loglike(data,D,K,pis,mus,sigmas):

    #calculate loglikelihood
    llh = [0]
    for n in range(N):
        for k in range(K):
            llh += np.log(pis[k] * multivariate_normal.pdf(data[n],mus[k],sigmas[k]))

    return llh


# EM main loop: sample data from GMM, iterate e- and m-steps
def em_main():

    data = sample_GMM()
    K = 4
    D = 2
    N = 500
    x = data[:,0]
    y = data[:,1]

    # initialize parameters
    init = initialize_uniformly(data,D,K)
    mus = init[0]
    sigmas = init[1]
    pis = init[2]

    # iterate e-step and m-step
    iter = 0
    #llh_new, llh_old = 0, 1e-8
    while iter < 30:
        iter +=1

        ### calculate loglikelihood
        #llh_old= loglike(data,D,K,pis,mus,sigmas)

        ### execute e_step
        resp = e_step(data,D,K,mus,sigmas,pis)

        ### execute m_step
        m = m_step(data,D,K,resp)
        mus = m[0]
        sigmas = m[1]
        pis = m[2]

        ### calculate loglikelihood
        #llh_new = loglike(data,D,K,pis,mus,sigmas)

    plot_data = plt.scatter(x,y,c='black', s=2)

    # Generate grid points
    dimension = 100
    x, y = np.meshgrid(np.linspace(-2,4,dimension),np.linspace(-2,5,dimension))
    xy = np.column_stack([x.flat, y.flat])

    # density values at the grid points
    Z_1 = multivariate_normal.pdf(xy, mus[0,:], sigmas[0,:]).reshape(x.shape)
    Z_2 = multivariate_normal.pdf(xy, mus[1,:], sigmas[1,:]).reshape(x.shape)
    Z_3 = multivariate_normal.pdf(xy, mus[2,:], sigmas[2,:]).reshape(x.shape)
    Z_4 = multivariate_normal.pdf(xy, mus[3,:], sigmas[3,:]).reshape(x.shape)

    # create and save plot
    fig = plt.contour(x, y, Z_1)
    fig = plt.contour(x, y, Z_2)
    fig = plt.contour(x, y, Z_3)
    fig = plt.contour(x, y, Z_4)
    plt.savefig('em_iter_30.png', dpi=300)

    return mus, sigmas

em_main()
```