Machine Learning

TECHNISCHE UNIVERSITÄT DARMSTADT

Summer Semester 2017, Homework 2 (90 points + 15 bonus)

Prof. Dr. J. Peters, M. Ewerton, S. Parisi
Wang Yujue 2573991, Zhi Rong 2806891

Due Date: Wednesday, 24 May 2017 (before the lecture)

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?

Answer:

Bayesian Decision Theory is method to find the a-posteriori probability of the class C_k given the observation (feature) x. The goal is to classify new "object" so that the probability of a wrong classification is minimized. The posteriori of the two classes should be equal at the optimal decision boundary. If $p(C_1|x) > p(C_2|x)$ or $p(x|C_1)p(C_1) > p(x|C_2)p(C_2)$, we will decide C_1 over C_2

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.

$$x^* = \frac{\mu_1+\mu_2}{2}$$

c) Different Misclassification Costs [8 Points]

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

Answer:

According to the risk minimization, we have a decision rule $\frac{p(x|C_1)}{p(x|C_2)} > \frac{\lambda_{12}-\lambda_{22}}{\lambda_{21}-\lambda_{11}} \cdot \frac{p(C_2)}{p(C_1)}$

Where under the condition of this problem, $\lambda_{21} = 4\lambda_{12}$, and $\lambda_{11} = \lambda_{22} = 0$

Therefore,

$$x^* = \frac{4\sigma^2 ln2 + 3\mu_2^2}{2\mu_2}$$

SOLUTIONS

Machine Learning - Homework 2

Name, Vorname: _____ Matrikelnummer: ␣␣␣␣␣␣␣␣

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 Maximization Likelihood Estimate [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 and at http://en.wikipedia.org/wiki/Matrix_calculus.)

Answer:

$$\begin{aligned}
\nabla_\mu log L(\theta) &= \nabla_\mu \log p(\boldsymbol{X}|\boldsymbol{\theta}) = \nabla_\mu \sum_{n=1}^{N} log p(\boldsymbol{x}_n|\boldsymbol{\mu},\boldsymbol{\Sigma}) \\
&= \nabla_\mu \sum_{n=1}^{N} log p(\boldsymbol{x}_n|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \sum_{n=1}^{N} \nabla_\mu log p(\boldsymbol{x}_n|\boldsymbol{\mu},\boldsymbol{\Sigma}) \\
&= \sum_{n=1}^{N} \nabla_\mu log \left(\frac{1}{\sqrt{2\pi\boldsymbol{\Sigma}}} exp\left(-\frac{1}{2}(\boldsymbol{x}_n-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}) \right) \right) \\
&= \sum_{n=1}^{N} \nabla_\mu \left(log(1) - log\sqrt{\boldsymbol{\Sigma}} - log\left(\sqrt{2\pi}\right) + log\left(exp\left(-\frac{1}{2}(\boldsymbol{x}_n-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}) \right) \right) \right) \\
&= \sum_{n=1}^{N} \nabla_\mu log \left(exp\left(-\frac{1}{2}(\boldsymbol{x}_n-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}) \right) \right) \\
&= \sum_{n=1}^{N} (\boldsymbol{x}_n-\boldsymbol{\mu})\boldsymbol{\Sigma}^{-1} = 0
\end{aligned} \tag{3}$$

Thus, $\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n$

$$\begin{aligned}
\nabla_\Sigma log L(\theta) &= \sum_{n=1}^{N} \nabla_\Sigma log \left(\frac{1}{\sqrt{2\pi\boldsymbol{\Sigma}}} exp\left(-\frac{1}{2}(\boldsymbol{x}_n-\boldsymbol{\mu})^T \boldsymbol{\Sigma}(\boldsymbol{x}_n-\boldsymbol{\mu}) \right) \right) \\
&= \sum_{n=1}^{N} \nabla_\Sigma \left(log(1) - log\sqrt{\boldsymbol{\Sigma}} - log\left(\sqrt{2\pi}\right) + log\left(exp\left(-\frac{1}{2}(\boldsymbol{x}_n-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}) \right) \right) \right) \\
&= \sum_{n=1}^{N} \nabla_\Sigma \left(log\left(exp\left(-\frac{1}{2}(\boldsymbol{x}_n-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}) \right) \right) - log\sqrt{\boldsymbol{\Sigma}} \right) \\
&= \sum_{n=1}^{N} \left((\boldsymbol{x}_n-\boldsymbol{\mu})(\boldsymbol{x}_n-\boldsymbol{\mu})^T - \boldsymbol{\Sigma} \right) = 0
\end{aligned} \tag{4}$$

Thus, $\hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{n=1}^{N} (\boldsymbol{x}_n-\hat{\boldsymbol{\mu}})(\boldsymbol{x}_n-\hat{\boldsymbol{\mu}})^T$

SOLUTIONS

b) Prior Probabilities [2 Points]

Compute the prior probability of each class from the dataset.

Answer:
$P_{C1} = \frac{\#ofC1}{\#ofalldata} = \frac{239}{239+761} = 0.239$
$P_{C2} = \frac{\#ofC2}{\#ofalldata} = \frac{761}{239+761} = 0.761$

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!

Answer: The bias of an estimator could be defined as the difference between this estimator's expected value and the true value of the parameter being estimated.
(code is attached in the end)

$$Bias = E[\hat{\theta}] - \theta \tag{6}$$

$\hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x_n} - \hat{\boldsymbol{\mu}})(\boldsymbol{x_n} - \hat{\boldsymbol{\mu}})^T$, this sample variance is uncorrected, which means it is a biased estimator of the variance. While, $\hat{\boldsymbol{\Sigma}} = \frac{1}{N-1}\sum_{n=1}^{N}(\boldsymbol{x_n} - \hat{\boldsymbol{\mu}})(\boldsymbol{x_n} - \hat{\boldsymbol{\mu}})^T$ is unbiased estimator of the variance. We need to calculate μ and covariance matrix Σ of two classes.

$$\mu_1 = (-0.707, -0.813), \Sigma_{biased1} = \begin{bmatrix} 9.020 & 2.673 \\ 2.673 & 3.596 \end{bmatrix}, \Sigma_{unbiased1} = \begin{bmatrix} 9.096 & 2.695 \\ 2.695 & 3.627 \end{bmatrix} \tag{7}$$

$$\mu_2 = (3.985, 3.984), \Sigma_{biased2} = \begin{bmatrix} 4.176 & 0.0276 \\ 0.028 & 2.753 \end{bmatrix}, \Sigma_{unbiased2} = \begin{bmatrix} 4.186 & 0.028 \\ 0.027 & 2.760 \end{bmatrix} \tag{8}$$

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.)

We can see the result from figure 4.

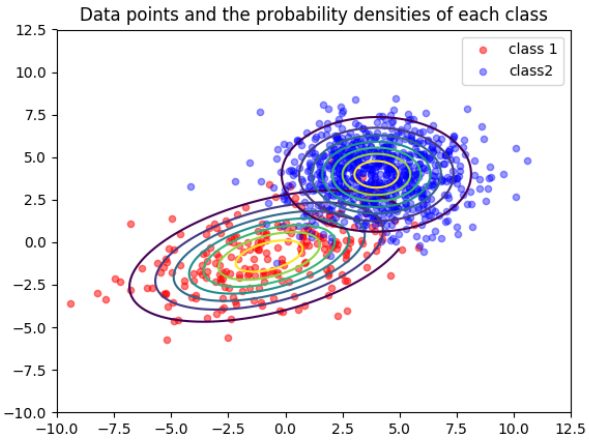
e) Posterior [8 Points]

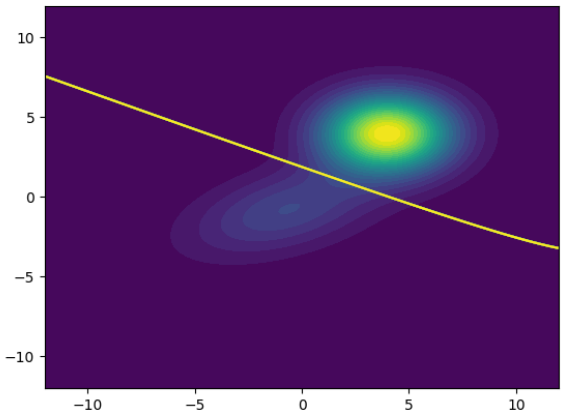
In a single graph, plot the posterior distribution of each class $p(C_i|x)$ and show the decision boundary.

The posterior distribution and the decision boundary is shown in figure 5

SOLUTIONS

Name, Vorname: _____ Matrikelnummer: ⌴⌴⌴⌴⌴⌴⌴





f) Bayesian Estimation [15 Bonus Points]

State the generic case of a Bayesian treatment of the parameters θ used to model the data, i.e., $p(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{Y})$. Derive the estimate of the mean, assuming that the model of the output variable is a Gaussian distribution with a fixed variance. Which are the advantages of being Bayesian?

$Y = X^T\theta + \epsilon$, with $E(\epsilon) = 0$ and X and ϵ independent.

ϵ are independent and identical normally distributed random variables: $\epsilon\ N\left(0,\sigma^2\right)$. $E(Y|x) = X^T\theta$.

The likelihood is Gaussian:

$$p\left(\boldsymbol{Y}|\boldsymbol{X},\theta,\sigma^2\right) = \frac{1}{(2\pi\sigma^2)^{n/2}} exp\left(-\frac{1}{2\sigma^2}||\boldsymbol{Y}-\boldsymbol{X}\theta||\right) \tag{12}$$

$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{X}^T\boldsymbol{X}^{-1}\right)\boldsymbol{X}^T\boldsymbol{Y}$ Then, the posterior distribution is:

$$p\left(\theta,\sigma^2|\boldsymbol{Y},\boldsymbol{X}\right) \propto exp\left(-\frac{1}{2\sigma^2}||\boldsymbol{Y}-\boldsymbol{X}\theta||^2 - 1/2\theta^T\Lambda\theta\right) \tag{13}$$

with $\theta \sim N(\mu_n, |\Sigma_n)$, where $\mu_n = \left(\boldsymbol{X}^T\boldsymbol{X} + \sigma^2\Lambda\right)^{-1}\boldsymbol{XY}$, $\Sigma_n = \sigma^2\left(\boldsymbol{X}^T\boldsymbol{X} + \sigma^2\Lambda\right)^{-1}$

Now, say we are given a new independent data point x, we want to predict the corresponding unseen dependent value y.

$$y|Y \sim N\left(x^T\mu_n, \sigma_n^2(x)\right), with\sigma_n^2(x) = \sigma^2 + x^T\Sigma_n x \tag{14}$$

The advantages of being Bayesian is you can: include prior knowledge and "learn" from existing evidence; easily extend to non-linear regression models; easily make inferences and predictions by including complex functions of prior model results; handle missing data directly in the model specification; directly report results in terms of probability.

SOLUTIONS

Name, Vorname: _____ Matrikelnummer: ⌴⌴⌴⌴⌴⌴⌴

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. 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.

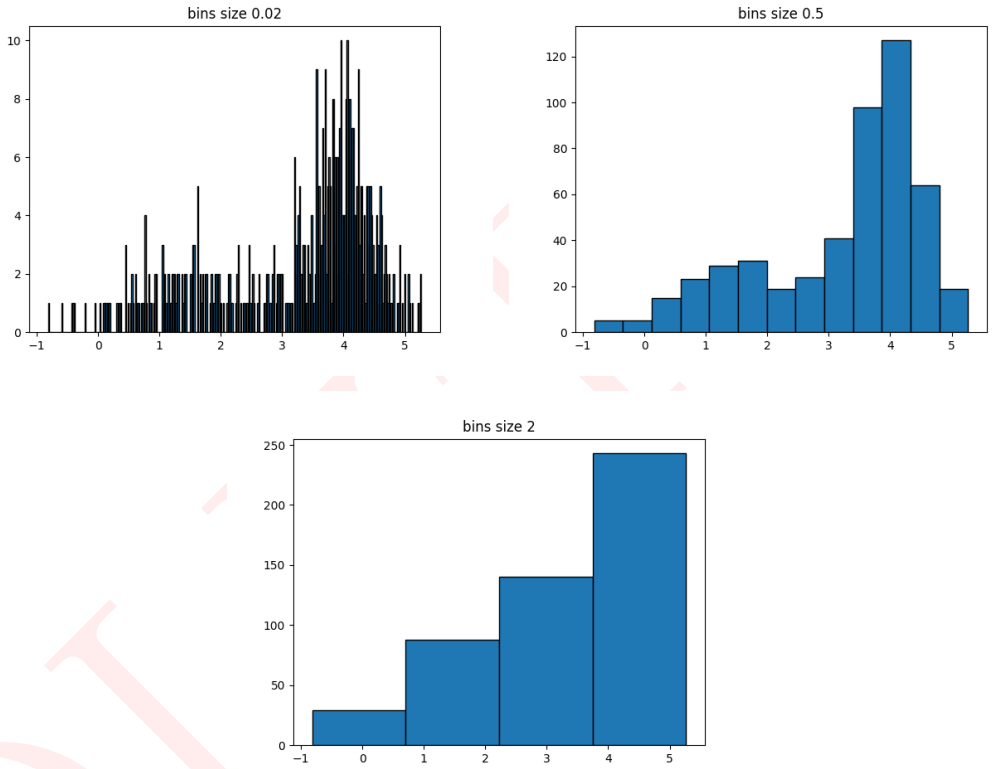


Figure 1: The histogram with different bin size.

Answer:

The 0.5 bin size performs best. Bin size 0.02 is too refined and noisy, it shows too much individual data and does not allow the underlying pattern (frequency distribution) of the data to be easily seen. While bin size 2.0 is too coarse, and contains too little information. We are unable to find the underlying trend in the data. Given only these three trials, bin size 0.5 performs the best.

The code is attached in the end.

b) Kernel Density Estimate [6 Points]

Compute the probability density estimate using a Gaussian kernel with $\sigma = 0.03$, $\sigma = 0.2$ and $\sigma = 0.8$. 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.

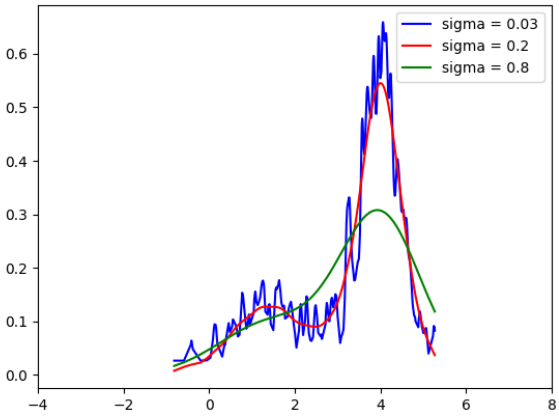


Figure 2: Gaussian kernel

Answer:

The log-likelihood for each case is:

$\sigma = 0.03, -674.73$;

$\sigma = 0.2, -717.02$;

$\sigma = 0.8, -795.66$

As we can see from figure 2, the curve with $\sigma = 0.2$ performs the best. It captures the main distribution of the data and is smooth. The curve with $\sigma = 0.2$ is most accurate curve(maximum log-likelihood) based on the training data set, but it is noisy not smooth enough. The curve with $\sigma = 0.8$ is too smooth, which cannot represent the original information from the dataset.

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. (Note: you need to normalize your solution according to K as the volume is not constant.)

As we can see from figure 3, the curve is noisy when k=2, the curve is about right when k=8, the curve is too smooth when k=35.

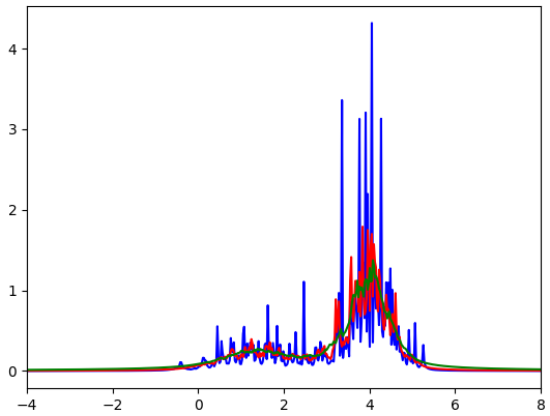


Figure 3: K-Nearest Neighbors

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?

We use training-set to train the parameters, and testing-set to estimate the error rate. An estimator performs good with training-set doesn't mean it also performs good with testing-set. That's why we need to test them on a different data-set.

The log-likelihoods of of the estimators are:

Test-set:

test-set: kernel density estimation: -inf -2904.3420759401097 -3188.8344841562775

test-set: k nearest neighbor: -1891.0653835430671 -1569.011937018638 -1460.1365086935637

training-set:

train-set: kernel density estimation: -674.7279387090637 -717.0216577444172 -795.663283345903

train-set: k nearest neighbor: -751.7696288034737 -631.4840782902313 -499.6239526040297

As we can see from the testing result, KDE estimators perform really bad at test-set. Thus, we will choose K-NN estimators.

SOLUTIONS

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.

Answer:

Parameters: Number of components, convergence threshold, number of EM iterations to perform, initial means, variance and probabilities.

Initialize: $\mu_1, \mu_2, \mu_3, \mu_4, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \pi_1, \pi_2, \pi_3, \pi_4$

Loop:
E-step: Compute posterior distributions for each mixture component and for all data points.

$$\alpha_j = p(j = j|x_n) = \frac{\pi_j N(x_n|\mu_j, \sigma_j)}{\sum_{i=1}^{M} \pi_i N(x_n|\mu_i, \sigma_i)}$$

M-step: Compute new weighted means of all data points.

$$\mu_j = \frac{1}{N_j} \sum_{n=1}^{N} \alpha_j x_n \text{ with } N_j = \sum_{n=1}^{N} \alpha_j$$

$$(\sigma_j)^2 = \frac{1}{N_j} \sum_{n=1}^{N} \alpha_j (x_n - \mu_j)^2$$

$$\pi_j^{New} = \frac{N_j}{N}$$

Repeat until convergence.

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.

Answer:

We can see the result from figure 4. The code is attached in the end.

SOLUTIONS

Name, Vorname: _____ Matrikelnummer: ⌴⌴⌴⌴⌴⌴⌴

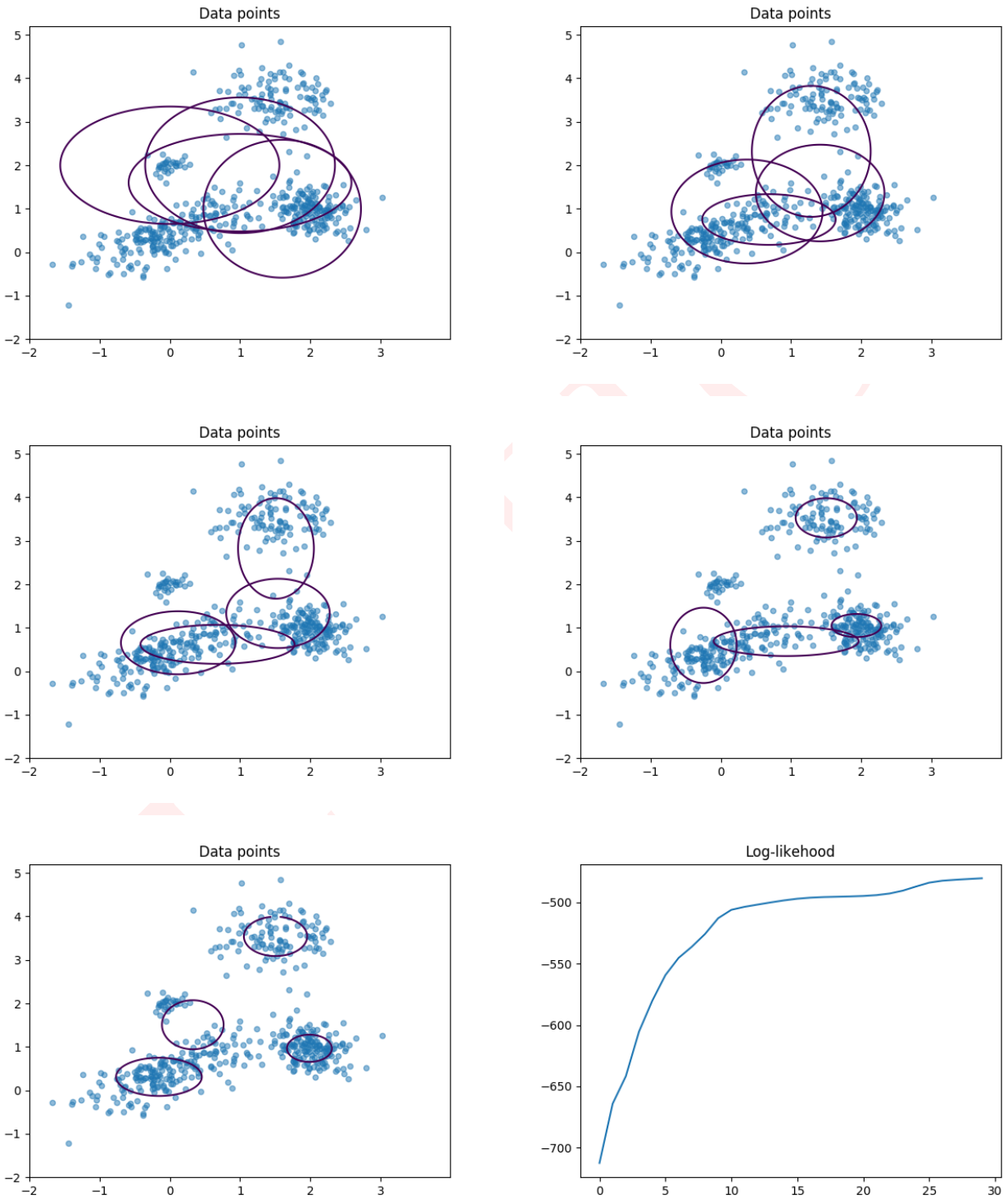


Figure 4: The Expectation Maximization and log-likelihood result.

Machine Learning - Homework 2

Name, Vorname: _____ Matrikelnummer: ␣␣␣␣␣␣␣␣

Code:

Problem 2.2

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

def read(path):
    data = []
    txt = open(path)
    for line in txt:
        x,y=map(float,line.strip().split())
        data.append((x,y))
    return np.asarray(data)

c1 = read("dataSets/densEst1.txt")
c2 = read("dataSets/densEst2.txt")

# calculate prior
prior_c1 = len(c1)/(len(c1)+len(c2))
prior_c2 = len(c2)/(len(c1)+len(c2))

# calculate mu and sigma
def estimate(data,biased=True):
    mu = np.mean(data,0)
    N = len(data)
    norm = N if biased else N-1
    sigma = 1/norm * sum((x-mu).reshape(2,1)*(x-mu) for x in data)
    return mu, sigma

mu_c1, sigma_biased_c1 = estimate(c1,biased=True)
_, sigma_unbiased_c1 = estimate(c1,biased=False)

mu_c2, sigma_biased_c2 = estimate(c2,biased=True)
_, sigma_unbiased_c2 = estimate(c2,biased=False)

print('Class 1: mu:',mu_c1,'Biased sigma:',sigma_biased_c1,'Unbiased sigma',sigma_unbiased_c1
print('Class 2: mu:',mu_c2,'Biased sigma:',sigma_biased_c2,'Unbiased sigma',sigma_unbiased_c2

# plot contour result
plt.figure()
x1,y1 = np.mgrid[-10:10:.01,-10:10:.01]
pos = np.empty(x1.shape + (2,))
pos[:, :, 0] = x1; pos[:, :, 1] = y1
z1= multivariate_normal(mu_c1,sigma_unbiased_c1)

x2,y2 = np.mgrid[-10:12.5:.01,-10:12.5:.01]
pos2 = np.empty(x2.shape + (2,))
pos2[:, :, 0] = x2; pos2[:, :, 1] = y2
z2= multivariate_normal(mu_c2,sigma_unbiased_c2)

c1=plt.scatter(c1[:,0],c1[:,-1],color='red',s=20,alpha=.5)
```

SOLUTIONS

```
c2=plt.scatter(c2[:,0],c2[:,-1],color='blue',s=20,alpha=.4)
plt.legend((c1,c2),('class 1','class 2'))
plt.contour(x1,y1,z1.pdf(pos))
plt.contour(x2,y2,z2.pdf(pos2))
plt.title('Data points and the probability densities of each class')
plt.show()

# plot posterior result
plt.figure()
x, y = np.mgrid[-12:12:.01,-12:12:.01]
posn = np.empty(x.shape + (2,))
posn[:, :, 0] = x; posn[:, :, 1] = y
d1 =  z1.pdf(posn)*prior_c1
d2 =  z2.pdf(posn)*prior_c2
plt.contourf(x,y,d1+d2,30)

# plot boundary
boundary = np.sign(d1-d2)
plt.contour(x,y,boundary)
plt.show()
```

Problem 2.3

```
import numpy as np
from math import log
from matplotlib import pyplot as plt



# 2.3.a
bin_size = 0.02

def read(path):
    data = []
    txt = open(path)
    for line in txt:
        data.append(float(line.strip()))
    return np.asarray(data)

train = read("dataSets/nonParamTrain.txt")
test = read("dataSets/nonParamTest.txt")

bins = int((max(train)-min(train))/bin_size+1)
plt.hist(train,bins=bins,edgecolor = 'black')
plt.title('bins size 0.02')
plt.show()


# 2.3.b
def kde(sigma,x1,x2):
    f = []
    for x0 in x2:
        sum = 0
```

SOLUTIONS

```
        for xn in x1:
            u = xn - x0
            K = np.exp(- u**2 /(2*sigma**2))/(np.sqrt(2*np.pi)*sigma)
            sum += 1/(len(x1)) * K
        f.append(sum)
    return f

def log_likelihood(f):
    loss = [log(fi) for fi in f]
    return sum(loss)


x_kde = sorted(train)
f1 = kde(0.03,train,x_kde)
f2 = kde(0.2,train,x_kde)
f3 = kde(0.8,train,x_kde)


plt.figure()
plt.plot(x_kde,f1,'b',label='sigma = 0.03')
plt.plot(x_kde,f2,'r',label='sigma = 0.2')
plt.plot(x_kde,f3,'g',label='sigma = 0.8')
plt.xlim([-4,8])
plt.legend()
plt.show()

# print(log_likelihood(f1),log_likelihood(f2),log_likelihood(f3))


# 2.3.c
def knn(data,x,k):
    result = []
    for xi in x:
        dist = [np.abs(point-xi) for point in data]
        V = np.sort(dist)[k]
        p = k/(V*len(data))
        result.append(p)
    return result

x_knn = np.linspace(-4,8,200)
knn_1 = knn(train,x_knn,2)
knn_2 = knn(train,x_knn,8)
knn_3 = knn(train,x_knn,35)


plt.figure()
plt.plot(x_knn,knn_1,'b',label='k=2')
plt.plot(x_knn,knn_2,'r',label='k=8')
plt.plot(x_knn,knn_3,'g',label='k=35')
plt.xlim([-4,8])
plt.show()
```

SOLUTIONS

```
test = sorted(test)

# 2.3.d
test_kde_1 = kde(0.03,train,test)
test_kde_2 = kde(0.2,train,test)
test_kde_3 = kde(0.8,train,test)

test_knn_1 = knn(train,test,2)
test_knn_2 = knn(train,test,8)
test_knn_3 = knn(train,test,35)

print('test-set: kernel density estimation:',log_likelihood(test_kde_2),log_likelihood(test_kde
print('test-set: k nearest neighbor:',log_likelihood(test_knn_1),log_likelihood(test_knn_2),log

print('train-set: kernel density estimation:',log_likelihood(f1),log_likelihood(f2),log_likeli
print('train-set: k nearest neighbor:',log_likelihood(knn_1),log_likelihood(knn_2),log_likeliho
```

Problem 2.4

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
from scipy.stats import multivariate_normal

filename = 'dataSets/gmm.txt'
x_file = []
y_file = []
with open(filename, 'r') as file_to_read:
    while True:
        lines = file_to_read.readline()
        if not lines:
            break
        x_tmp, y_tmp = [float(i) for i in lines.split()]
        x_file.append(x_tmp)
        y_file.append(y_tmp)

x_file = np.array(x_file)
y_file = np.array(y_file)


# Initialize with parameters roughly getting from the plot
# mu_1 = [0,0]
mu_1 = [1,2]
cov_1 = np.array([[1.5, 0], [0, 2]])
pi_1 = 1/4
mu_2 = [1,2]
cov_2 = np.array([[2, 0], [0, 1]])
pi_2 = 1/4
mu_3 = [1,1.6]
cov_3 = np.array([[2, 0], [0, 1]])
pi_3 = 1/4
mu_4 = [0,2]
cov_4 = np.array([[1, 0], [0, 2]])
```

SOLUTIONS

```
pi_4 = 1/4


def gaussian_2d(x, y, mu, cov):
    return mlab.bivariate_normal(x,y,np.sqrt(cov[0,0]),np.sqrt(cov[1,1]),mu[0],mu[1],np.sqrt(


likelihood = []
for i in range(30):
    i = i + 1

    gaussian_1 = gaussian_2d(x_file,y_file, mu_1, cov_1)
    gaussian_2 = gaussian_2d(x_file,y_file, mu_2, cov_2)
    gaussian_3 = gaussian_2d(x_file,y_file, mu_3, cov_3)
    gaussian_4 = gaussian_2d(x_file,y_file, mu_4, cov_4)

    p_y = pi_1 * gaussian_1 + pi_2 * gaussian_2 + pi_3 * gaussian_3 + pi_4 * gaussian_4
    q1 = (pi_1 * gaussian_1) / p_y
    q2 = (pi_2 * gaussian_2) / p_y
    q3 = (pi_3 * gaussian_3) / p_y
    q4 = (pi_4 * gaussian_4) / p_y

    mu_1 = [np.sum(q1 * x_file), np.sum(q1 * y_file)] / np.sum(q1)
    # sigxy1 = np.sum(q1*(x_file - mu_1[0])* ((y_file - mu_1[1])))
    sigxy1 = 0
    cov_1 = [[np.sum(q1 * ((x_file - mu_1[0]) ** 2)), sigxy1], [sigxy1, np.sum(q1 * ((y_file
    pi_1 = np.sum(q1) / 500
    # sigxy2 = np.sum(q1*(x_file - mu_2[0])* ((y_file - mu_2[1])))
    sigxy2 = 0
    mu_2 = [np.sum(q2 * x_file), np.sum(q2 * y_file)] / np.sum(q2)
    cov_2 = [[np.sum(q2 * ((x_file - mu_2[0]) ** 2)), sigxy2], [sigxy2, np.sum(q2 * ((y_file
    pi_2 = np.sum(q2) / 500
    # sigxy3 = np.sum(q1 * (x_file - mu_3[0]) * ((y_file - mu_3[1])))
    sigxy3 = 0
    mu_3 = [np.sum(q3 * x_file), np.sum(q3 * y_file)] / np.sum(q3)
    cov_3 = [[np.sum(q3 * ((x_file - mu_3[0]) ** 2)), sigxy3], [sigxy3, np.sum(q3 * ((y_file
    pi_3 = np.sum(q3) / 500
    # sigxy4 = np.sum(q1*(x_file - mu_4[0])* ((y_file - mu_4[1])))
    sigxy4 = 0
    mu_4 = [np.sum(q4 * x_file), np.sum(q4 * y_file)] / np.sum(q4)
    cov_4 = [[np.sum(q4 * ((x_file - mu_4[0]) ** 2)), sigxy4], [sigxy4, np.sum(q4 * ((y_file
    pi_4 = np.sum(q4) / 500

    logll = np.sum(np.log10(pi_1 * gaussian_1 + pi_2 * gaussian_2 + pi_3 * gaussian_3 + pi_4
    likelihood.append(logll)

x = np.linspace(-2,4)
y = np.linspace(-2,4)
X,Y = np.meshgrid(x, y)

# plot contour result
plt.figure()
```

SOLUTIONS

Name, Vorname: _____ Matrikelnummer: ␣␣␣␣␣␣␣␣

```
x1,y1 = np.mgrid[-2:4:.01,-2:4:.01]
pos = np.empty(x1.shape + (2,))
pos[:, :, 0] = x1; pos[:, :, 1] = y1
z1= multivariate_normal(mu_1,cov_1)

x2,y2 = np.mgrid[-2:4:.01,-2:4:.01]
pos2 = np.empty(x2.shape + (2,))
pos2[:, :, 0] = x2; pos2[:, :, 1] = y2
z2= multivariate_normal(mu_2,cov_2)

x3,y3 = np.mgrid[-2:4:.01,-2:4:.01]
pos3 = np.empty(x3.shape + (2,))
pos3[:, :, 0] = x3; pos3[:, :, 1] = y3
z3= multivariate_normal(mu_3,cov_3)


x4,y4 = np.mgrid[-2:4:.01,-2:4:.01]
pos4 = np.empty(x4.shape + (2,))
pos4[:, :, 0] = x4; pos4[:, :, 1] = y4
z4= multivariate_normal(mu_4,cov_4)

c1=plt.scatter(x_file,y_file,s=20,alpha=.5)
plt.contour(x1,y1,z1.pdf(pos),1)
plt.contour(x2,y2,z2.pdf(pos2),1)
plt.contour(x3,y3,z3.pdf(pos3),1)
plt.contour(x4,y4,z4.pdf(pos4),1)
plt.title('Data␣points')
plt.show()

plt.figure()
plt.plot(likelihood)
plt.title("Log-likehood")
plt.show()
```

SOLUTIONS