Statistical Machine Learning - Exercise 2

TECHNISCHE UNIVERSITÄT DARMSTADT

Dominik Marino - 2468378, Pertami Kunz - 2380210
8. August 2020

Inhaltsverzeichnis

1 Bayesian Decision Theory

Consider data generated by a mixture of two Gaussian distributions,

$$C_1 : X \sim \mathcal{N}\left(\mu_1, \sigma_1^2\right)$$
$$C_2 : X \sim \mathcal{N}\left(\mu_2, \sigma_2^2\right).$$

1.1 Optimal Boundary

Tutor solution:
Bayesian Decision Theory is a framework for taking optimal decisions under uncertain conditions (see Bishop ch. 1.5). The goal is to minimize the cost of classification.

Let R_1 and R_2 denote regions and let C_1 and C_2 denote the repective class to which a point x is assigned. In the case of two classes, the missclassification error is:

$$\begin{aligned} p(error) &= p(x \in R_1, C_2) + p(x \in R_2, C_1) \\ &= \int_{R_1} p(x, C_2) + \int_{R_2} p(x, C_1) \\ &= \int_{R_1} p(x \mid C_2)p(C_2) + \int_{R_2} p(x \mid C_1)p(C_1) \end{aligned}$$

Bayesian decision theory is about finding the boundary (threshold) to decide which distribution a set of observations belongs to, based on Bayes' theorem on a-posteriori probability;

$$p\left(C_k|\boldsymbol{x}\right) = \frac{p\left(\boldsymbol{x}|C_k\right)p\left(C_k\right)}{p\left(\boldsymbol{x}\right)},$$

where $p\left(\boldsymbol{x}|C_k\right)$ is likelihood, $p\left(C_k\right)$ is the class prior, and $p\left(\boldsymbol{x}\right)$ is the normalization term. The goal is to minimize the misclassification rate. For two classes C_1 and C_2, at the optimal decision boundary,

$$p\left(C_1|\boldsymbol{x}\right) = p\left(C_2|\boldsymbol{x}\right).$$

We decide for class C_1 over C_2 when $p\left(C_1|\boldsymbol{x}\right) > p\left(C_2|\boldsymbol{x}\right)$, that is if class C_1 has th highest a-posteriori probability.

1.2 Decision Boundaries

At the decision boundary with $p\left(C_1\right) = p\left(C_2\right)$, $\sigma_1^2 = \sigma_2^2 = \sigma^2$ (and same misclassification costs):

$$\begin{aligned} p\left(C_1|\boldsymbol{x}\right) &= p\left(C_2|\boldsymbol{x}\right) \\ \frac{p\left(\boldsymbol{x}|C_1\right)\cancel{p\left(C_1\right)}}{\cancel{p\left(\boldsymbol{x}\right)}} &= \frac{p\left(\boldsymbol{x}|C_2\right)\cancel{p\left(C_2\right)}}{\cancel{p\left(\boldsymbol{x}\right)}} \\ \cancel{\frac{1}{\sqrt{2\pi\sigma^2}}} \exp\left(-\frac{1}{2\sigma^2}\left(x-\mu_1\right)^2\right) &= \cancel{\frac{1}{\sqrt{2\pi\sigma^2}}} \exp\left(-\frac{1}{2\sigma^2}\left(x-\mu_2\right)^2\right) \\ \exp\left(-\frac{1}{2\sigma^2}\left(x-\mu_1\right)^2 + \frac{1}{2\sigma^2}\left(x-\mu_2\right)^2\right) &= 1 \\ \Leftrightarrow -\left(x-\mu_1\right)^2 + \left(x-\mu_2\right)^2 &= 0 \\ 2x\mu_1 - \mu_1^2 - 2x\mu_2 + \mu_2^2 &= 0 \\ \Leftrightarrow 2x\left(\mu_1 - \mu_2\right) &= \mu_1^2 - \mu_2^2 \\ \Leftrightarrow x &= \frac{\mu_1^2 - \mu_2^2}{2\left(\mu_1 - \mu_2\right)} = \frac{1}{2}\left(\mu_1 + \mu_2\right). \end{aligned}$$

1.3 Different Misclassification Costs

Loss Matrix:

$$\lambda = \begin{pmatrix} \lambda_{1,1} & \lambda_{1,2} \\ \lambda_{2,1} & \lambda_{2,2} \end{pmatrix} = \begin{pmatrix} \lambda(\alpha_1 \mid C_1) & \lambda(\alpha_1 \mid C_2) \\ \lambda(\alpha_2 \mid C_1) & \lambda(\alpha_2 \mid C_2) \end{pmatrix} = \begin{pmatrix} C_1C_1 & C_1C_2 \\ C_2C_1 & C_2C_2 \end{pmatrix} = \begin{pmatrix} 0 & 4 \\ 1 & 0 \end{pmatrix} \tag{1}$$

with:

$$R(\alpha_i \mid x) = \sum_j \lambda(\alpha_i \mid C_j) p(C_i \mid x)$$

where also:

$$\lambda(\alpha_i \mid C_j) = \lambda_{i,j}$$

Decide α_1, if $R(\alpha_2 \mid x) > R(\alpha_1 \mid x)$

$$\begin{aligned} \lambda_{2,1} p(C_1 \mid x) + \lambda_{2,2} p(C_2 \mid x) &> \lambda_{1,1} p(C_1 \mid x) + \lambda_{1,2} p(C_2 \mid x) \\ \lambda_{2,1} p(C_1 \mid x) + \cancelto{0}{\lambda_{2,2}} p(C_2 \mid x) &> \cancelto{0}{\lambda_{1,1}} p(C_1 \mid x) + \lambda_{1,2} p(C_2 \mid x) \\ \lambda_{2,1} p(C_1 \mid x) &> \lambda_{1,2} p(C_2 \mid x) \\ p(C_1 \mid x) &> 4 * p(C_2 \mid x) \end{aligned}$$

With Bayes:

$$\begin{aligned} \frac{P(x \mid C_1) * P(C_1)}{P(x)} &> 4 * \frac{P(x \mid C_2 * P(C_2))}{P(x)} \\ \frac{P(x \mid C_1) * P(C_1)}{\cancel{P(x)}} &> 4 * \frac{P(x \mid C_2) * P(C_2)}{\cancel{P(x)}} \\ P(x \mid C_1) * \cancel{P(C_1)} &> 4 * P(x \mid C_2) * \cancel{P(C_2)} \\ P(x \mid C_1) &> 4 * P(x \mid C_2) \end{aligned}$$

With Gaussian:

$$p(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

Decision Boundary at p(C_1) = p(C_2), $\sigma_1^2 = \sigma_2^2 = \sigma^2$, $\mu_1 > 0, \mu_1 = 2\mu_2$ (and same misclassification costs):

$$\begin{aligned} p(C_1 \mid x) &> 4 * p(C_2 \mid x) \\ \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(x-\mu_1)^2}{2\sigma^2}) &> 4 * \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(x-\mu_2)^2}{2\sigma^2}) \\ \cancel{\frac{1}{\sqrt{2\pi\sigma^2}}} \exp(-\frac{(x-\mu_1)^2}{2\sigma^2}) &> 4 * \cancel{\frac{1}{\sqrt{2\pi\sigma^2}}} \exp(-\frac{(x-\mu_2)^2}{2\sigma^2}) \\ \exp(-\frac{(x-\mu_2)^2}{2\sigma^2}) &> 4 * \exp(-\frac{(x-\mu_2)^2}{2\sigma^2}) \\ -\frac{(x-\mu_2)^2}{2\sigma^2} &> \ln(4) - \frac{(x-\mu_1)^2}{2\sigma^2} \\ -(x-\mu_2)^2 &> 2\sigma_2^2 \ln(4) - (x-\mu_1)^2 \\ -x^2 + 2x\mu_1 - \mu_1^2 &> 2\sigma_2^2 \ln(4) - (x^2 - 2x\mu_2 - \mu_2^2) \\ 2(\mu_1 - \mu_2)x &> 2\sigma_2^2 ln(4) + \mu_1^2 - \mu_2^2 \\ x &> \frac{2\sigma_2^2 \ln(4) + \mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} \end{aligned}$$

with $\mu_1 = 2\mu_2$

$$x > \frac{2\sigma^2 \ln(4) + 3\mu_2^2}{2\mu_2}$$

When the cost of deciding C_1 when the true class is C_2 is higher than the other way round, make sense that the decision boundary is now shifted to the right closer to μ_1 (in this case by $\frac{\sigma^2 \cdot \ln(4)}{\mu_2}$), because now we are more conservative in choosing C_1.

2 Density Estimation

2.1 Gaussian Maximization Likelihood Estimate

The p-variate Gaussian distribution, $\boldsymbol{x} \sim \mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with $\boldsymbol{x}, \boldsymbol{\mu} \in \mathbb{R}^p, \boldsymbol{\Sigma} \in \mathbb{R}^{p \times p}$:

$$f(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{p}{2}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right],$$

where $|\boldsymbol{\Sigma}|$ is the determinant of the covariance matrix

With a set of i.i.d data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ drawn from $\mathcal{N}_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, the likelihood function is

$$\ell = p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{Np}{2}} |\boldsymbol{\Sigma}|^{-\frac{N}{2}} \exp\left[-\frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{x}_n-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu})\right],$$

and the log likelihood

$$\mathcal{L} = \ln \ell = -\frac{N}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{x}_n-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}) + [\text{some constant}].$$

Note that the square mahalanobis distance $D^2 = (\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) = \text{Tr}\left[\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})^T(\boldsymbol{x}-\boldsymbol{\mu})\right]$,where Tr $[\cdot]$ is the trace function.

- Taking derivative w.r.t μ and setting it to zero:

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\mu}}\left\{-\frac{N}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}\text{Tr}\left[\boldsymbol{\Sigma}^{-1}\sum_{n=1}^{N}(\boldsymbol{x}_n-\boldsymbol{\mu})^T(\boldsymbol{x}_n-\boldsymbol{\mu})\right]\right\} &\overset{!}{=} 0 \\
\cancel{\frac{1}{2}}\left\{\cancel{-2}\boldsymbol{\Sigma}^{-1}\sum_{n=1}^{N}(\boldsymbol{x}_n-\boldsymbol{\mu})\right\} &\overset{!}{=} 0 \\
\sum_{n=1}^{N}\boldsymbol{x}_n - N\boldsymbol{\mu} &\overset{!}{=} 0 \\
\Rightarrow \hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n.&
\end{aligned}$$

- Taking derivative w.r.t $\boldsymbol{\Sigma}^{-1}$ and setting it to zero:

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\Sigma}^{-1}}\left\{-\frac{N}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}\text{Tr}\left[\boldsymbol{\Sigma}^{-1}\sum_{n=1}^{N}(\boldsymbol{x}_n-\boldsymbol{\mu})^T(\boldsymbol{x}_n-\boldsymbol{\mu})\right]\right\} &\overset{!}{=} 0 \\
\frac{\partial}{\partial \boldsymbol{\Sigma}^{-1}}\left\{N\ln\left|\boldsymbol{\Sigma}^{-1}\right| - \text{Tr}\left[\boldsymbol{\Sigma}^{-1}\sum_{n=1}^{N}(\boldsymbol{x}_n-\boldsymbol{\mu})^T(\boldsymbol{x}_n-\boldsymbol{\mu})\right]\right\} &\overset{!}{=} 0 \\
N\boldsymbol{\Sigma}^T - \sum_{n=1}^{N}(\boldsymbol{x}_n-\boldsymbol{\mu})(\boldsymbol{x}_n-\boldsymbol{\mu})^T &\overset{!}{=} 0 \\
\Rightarrow \hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n-\hat{\boldsymbol{\mu}})(\boldsymbol{x}_n-\hat{\boldsymbol{\mu}})^T.&
\end{aligned}$$

  Using:
  - $|\boldsymbol{\Sigma}|^{-1} = \left|\boldsymbol{\Sigma}^{-1}\right|$
  - $\frac{\partial}{\partial \boldsymbol{A}}\text{Tr}[\boldsymbol{AB}] = \boldsymbol{B}^T$
  - $\frac{\partial}{\partial \boldsymbol{A}}\ln|\boldsymbol{A}| = \left(\boldsymbol{A}^{-1}\right)^T$
  - $\boldsymbol{\Sigma}^T = \boldsymbol{\Sigma}$.

2.2 Prior Probabilities

Probability Rule:

$$\sum P(X_n) = \sum_n P(X_n \in C_i) = 1, \forall i$$

Formula:

$$P(C_i) = \frac{|C_i|}{\sum_i |C_i|}$$

Prior Probability of densEst1	Prior Probability of densEst2
0.239	0.761

where C_i is the cardinality of class i

2.3 Biased ML Estimate

Tutor solution

The bias of an estimatro $\hat{\theta}$ is the drift of the estimated value form the true value. It can be comtuted by:

$$bias(\hat{\theta} = \mathbb{E}_\theta[\hat{\theta} - \theta]$$

where $\mathbb{E}_\theta$ denotes the expected value over the distribution $p_\theta(x)$.

Bias of an estimator is the difference between this estimator's expected value and the true value of the parameter being estimated.

	Definition of Biased Estimator	Definition of Unbiased Estimator
sample variance	$\hat{\sigma} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}})^2$	$\hat{\sigma} = \frac{1}{N-1}\sum_{n=1}^{N}(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}})^2$
sample mean	$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n$	$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n$

Multivariate Gaussian Distribution:

$$f(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{p}{2}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right],$$

Parameters:

- $\boldsymbol{\mu}$: mean –> have to be calculated
- $\boldsymbol{\Sigma}$: Covariance –> have to be calculated
- $p = 2$: dimension –> given
- $\boldsymbol{x}$: datapoints –> given
- π: constant

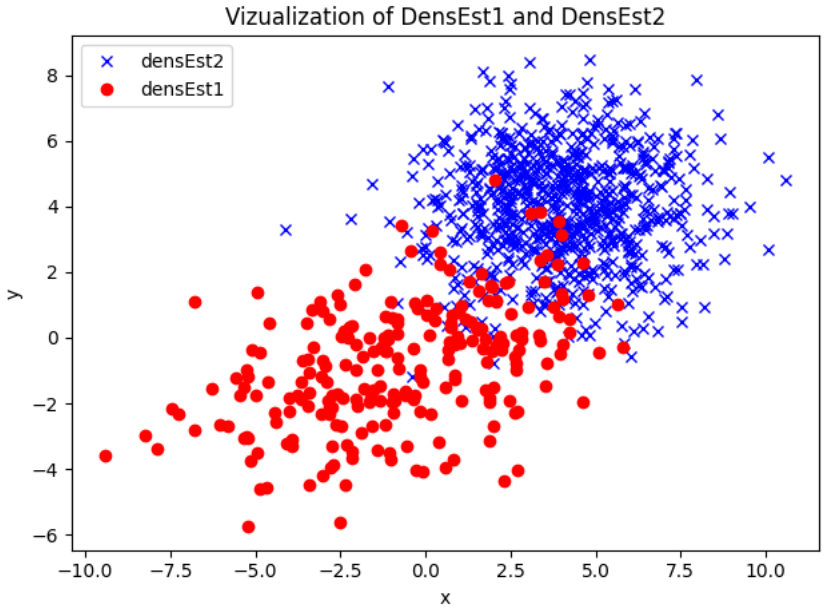
Then

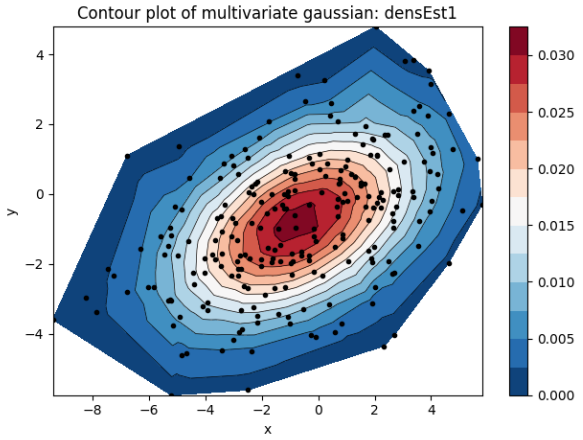
$$f(\boldsymbol{x}|C_i) = f\left(\boldsymbol{x}|\hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\Sigma}}_i\right) = (2\pi)^{-\frac{p}{2}} \left|\hat{\boldsymbol{\Sigma}}_i\right|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(\boldsymbol{x} - \hat{\boldsymbol{\mu}}_i)^T \hat{\boldsymbol{\Sigma}}_i^{-1}(\boldsymbol{x} - \hat{\boldsymbol{\mu}}_i)\right]$$

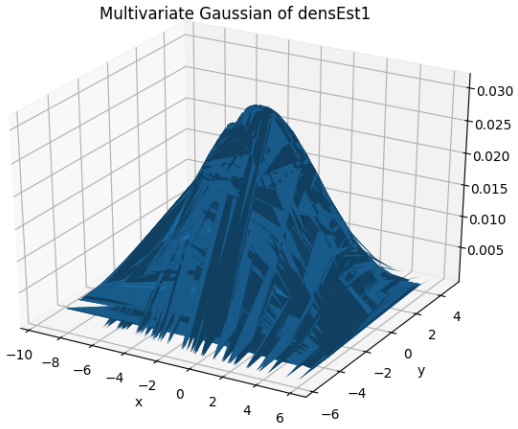
	DensEst1	DensEst2
Mean	(-0.70681374, -0.81343083)	(3.98534252, 3.98438364)
Covariance Unbiased Estimator	$\begin{pmatrix} 9.05742302 & 2.6841014 \\ 2.6841014 & 3.61145033 \end{pmatrix}$	$\begin{pmatrix} 4.18087542 & 0.02761954 \\ 0.02761954 & 2.75658555 \end{pmatrix}$
Covariance biased estimator	$\begin{pmatrix} 9.01952586 & 2.67287085 \\ 2.67287085 & 3.59633965 \end{pmatrix}$	$\begin{pmatrix} 4.1753815 & 0.02758324 \\ 0.02758324 & 2.75296323 \end{pmatrix}$

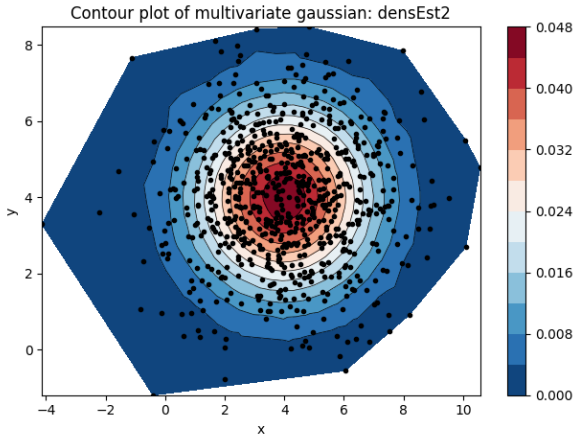
Tutor has different numbers

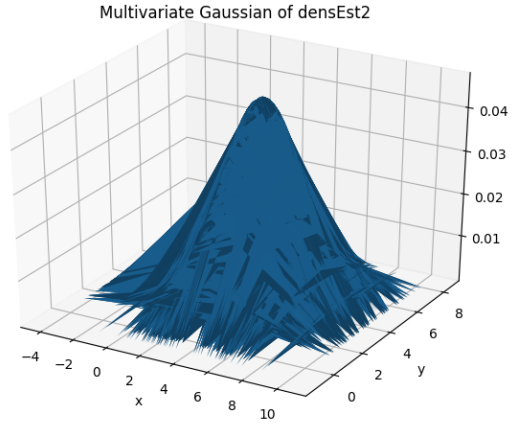
2.4 Class Density



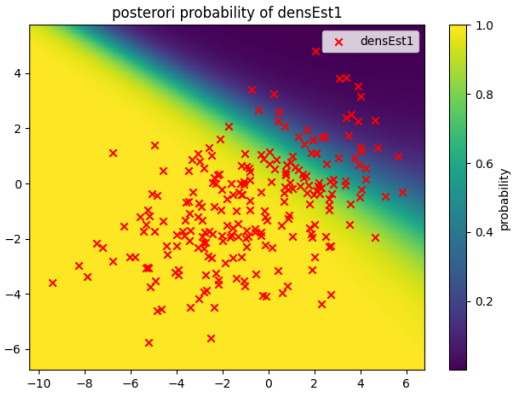


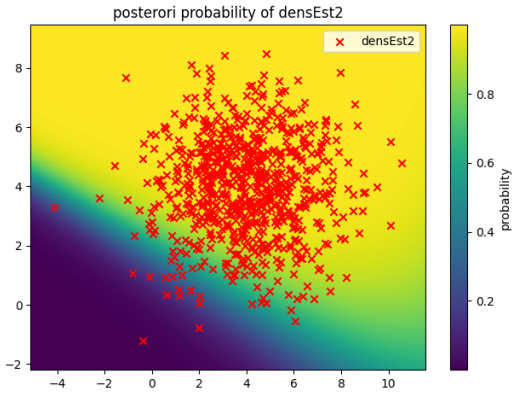


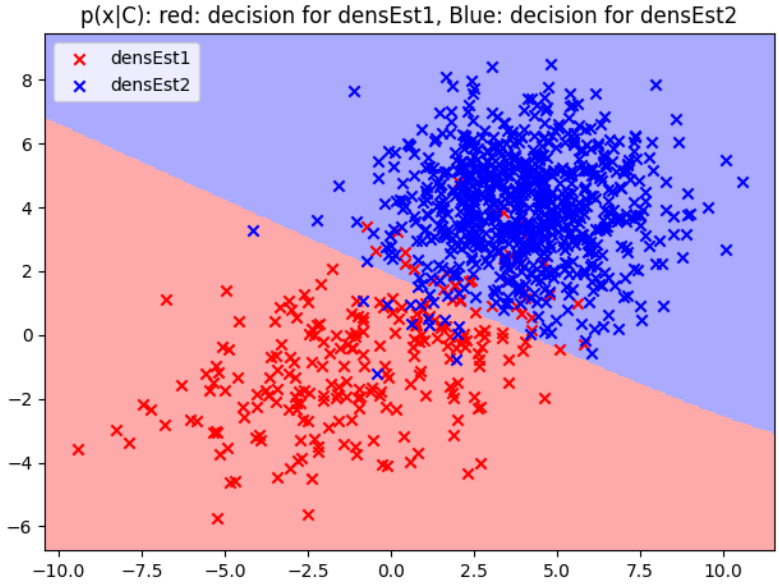




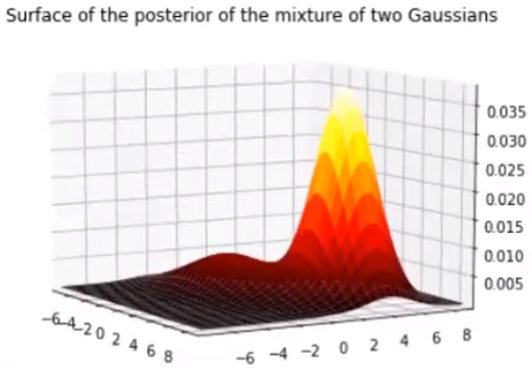
2.5 Posterior

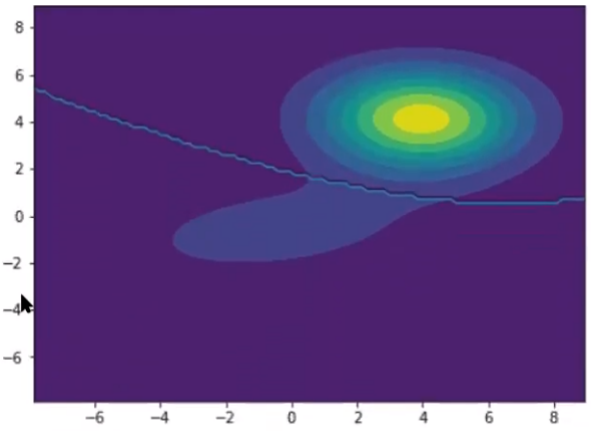


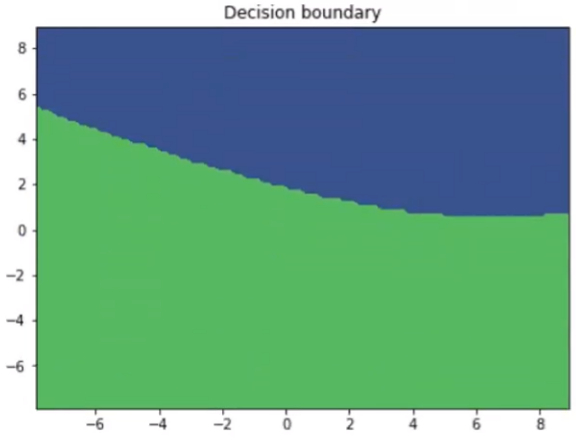


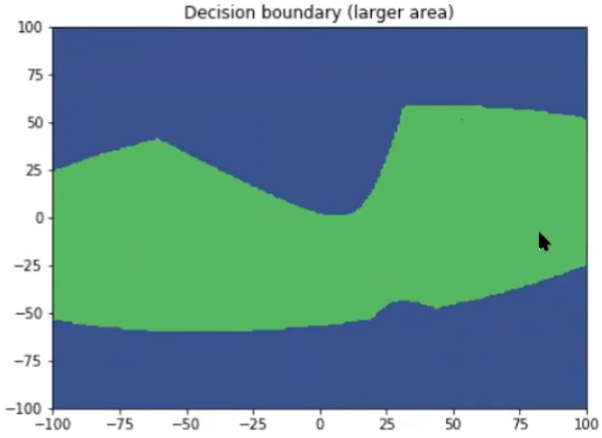


Tutors solution plot:










```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

""" 2c) Biased ML Estimate """
class Data:
    mean = 0

    variance_unbiased = 0
    variance_biased = 0

    cov_biased = 0
    cov_unbiased = 0

    def __init__(self, x, y, name):
        self.x = x
        self.y = y
        self.name = name

        self.calc_mean()
        self.calc_sample_variance_biased_and_unbiased()
        self.covariance_setup()

    def calc_covariance_biased(self, x, y):
        if(x == 0 and y == 0):
            return self.variance_biased[x]
        elif(x == 1 and y == 1):
            return self.variance_biased[y]
        else:
            sum = 0
            for i in range(self.x.shape[0]):
                sum += (self.x[i] - self.mean[0])*(self.y[i] - self.mean[1])
            return sum / self.x.shape[0]


    def calc_covariance_unbiased(self, x, y):
        if (x == 0 and y == 0):
            return self.variance_unbiased[x]
        elif (x == 1 and y == 1):
            return self.variance_unbiased[y]
        else:
            sum = 0
            for i in range(self.x.shape[0]):
                sum += (self.x[i] - self.mean[0])*(self.y[i] - self.mean[1])
            return sum / (self.x.shape[0] - 1)

    def covariance_setup(self):
        self.cov_biased = np.asarray([self.calc_covariance_biased(x, y) for x in range(2) for y in range(2)]).reshape(2,
     2)
        self.cov_unbiased = np.asarray([self.calc_covariance_unbiased(x, y) for x in range(2) for y in range(2)]).reshape
     (2, 2)

    def calc_mean(self):
        val_x = 0
        val_y = 0
        for i in range(self.x.shape[0]):
            val_x += self.x[i]
```

```
            val_y += self.y[i]
        self.mean = np.asarray([val_x / len(self.x), val_y / len(self.y)])

    def calc_sample_variance_biased_and_unbiased(self):
        val_x = 0
        val_y = 0
        for i in range(self.x.shape[0]):
            val_x += (self.x[i] - self.mean[0])**2
            val_y += (self.y[i] - self.mean[1])**2

        self.variance_unbiased = np.asarray([val_x / (len(self.x) - 1), val_y / (len(self.y) - 1)])
        self.variance_biased = np.asarray([val_x / len(self.x), val_y / len(self.y)])
""" end 2c) """


def get_data(name :str) -> np.asarray:
    file = open(name, 'r')
    x = []
    y = []
    for i in file.readlines():
        line = i.split()
        x.append(float(line[0]))
        y.append(float(line[1]))
    file.close()
    return np.asarray(x), np.asarray(y)

def show_data(densEst1, densEst2):
    plt.figure()
    plt.plot(densEst2.x, densEst2.y, 'bo', marker='x', label='densEst2')
    plt.plot(densEst1.x, densEst1.y, 'ro', marker='o', label='densEst1')
    plt.title('Vizualization of DensEst1 and DensEst2')
    plt.legend()
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

def calc_multi_gaussian(x, cov, mean):
    factor = 1 / np.sqrt((2*np.pi)**2 * np.linalg.det(cov))
    z = []
    for pos in x:
        multi = factor * np.exp(-0.5 * (pos - mean) @ np.linalg.inv(cov) @ (pos.T - mean.T))
        z.append(multi)
    return np.asarray(z)

def multi_gaussian(data):
    z = calc_multi_gaussian(np.c_[data.x, data.y], data.cov_unbiased, data.mean)
    fig, ax2 = plt.subplots()
    ax2.tricontour(data.x, data.y, z, levels=14, linewidths=0.5, colors='k')
    cntr2 = ax2.tricontourf(data.x, data.y, z, levels=14, cmap="RdBu_r")

    fig.colorbar(cntr2, ax=ax2)
    ax2.plot(data.x, data.y, 'ko', ms=3)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Contour plot of multivariate gaussian: ' +  data.name)
    plt.subplots_adjust(hspace=0.5)
    plt.show()

def prior_probability(densEst1, densEst2):
    prior_dens1 = len(densEst1.x) / (len(densEst1.x) + len(densEst2.x))
    prior_dens2 = len(densEst2.x) / (len(densEst1.x) + len(densEst2.x))
    return np.asarray([prior_dens1, prior_dens2])

def calculate_posterori(densEst1, densEst2, xx, yy):
    def likelihood(xx, yy, cov, mean):
        likeli = np.zeros(xx.shape)
        for i in range(xx.shape[1]):
            likeli[:, i] = calc_multi_gaussian(np.c_[xx[:, i], yy[:, i]], cov, mean)
        return likeli

    prior_dens1, prior_dens2 = prior_probability(densEst1, densEst2)

    likelihood_dens1 = likelihood(xx, yy, densEst1.cov_unbiased, densEst1.mean)
    likelihood_dens2 = likelihood(xx, yy, densEst2.cov_unbiased, densEst2.mean)
    normalization = likelihood_dens1 * prior_dens1 + likelihood_dens2 * prior_dens2

    posterori_dens1 = likelihood_dens1 * prior_dens1 / normalization
    posterori_dens2 = likelihood_dens2 * prior_dens2 / normalization


```

```
    return np.asarray(posterori_dens1), np.asarray(posterori_dens2)

def decision(Z1, Z2):
    Z = np.zeros(Z1.shape)
    for i in range(Z1.shape[0]):
        for j in range(Z1.shape[1]):
            if(Z1[i, j] >= Z2[i, j]):
                Z[i, j] = 0
            else:
                Z[i, j] = 1
    return Z


def multi_single_plot(densEst1, densEst2):
    cmap_light = ListedColormap(["#FFAAAA", "#AAAAFF"])
    X = np.c_[densEst1.x, densEst1.y]
    x_min1, x_max1 = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min1, y_max1 = X[:, 1].min() - 1, X[:, 1].max() + 1

    X = np.c_[densEst2.x, densEst2.y]
    x_min2, x_max2 = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min2, y_max2 = X[:, 1].min() - 1, X[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x_min1, x_max1, 0.05), np.arange(y_min1, y_max1, 0.05))
    Z, _ = calculate_posterori(densEst1, densEst2, xx, yy)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap='viridis')
    plt.colorbar(label='probability')
    plt.title('posterori probability of ' + densEst1.name)
    # Plot also the training points
    plt.scatter(densEst1.x, densEst1.y, marker='x', color='red', label=densEst1.name)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.legend()
    plt.show()


    xx, yy = np.meshgrid(np.arange(x_min2, x_max2, 0.05), np.arange(y_min2, y_max2, 0.05))
    _, Z = calculate_posterori(densEst1, densEst2, xx, yy)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap='viridis')
    plt.colorbar(label='probability')
    plt.title('posterori probability of ' + densEst2.name)
    # Plot also the training points
    plt.scatter(densEst2.x, densEst2.y, marker='x', color='red', label=densEst2.name)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.legend()
    plt.show()

    """Plot for decision Boundary """
    x_min = min(x_min1, x_min2)     #find min element (x_value) of densEst1 and densEst2
    x_max = max(x_max1, x_max2)     #find max element (x_value) " " " "
    y_min = min(y_min1, y_max1)
    y_max = max(y_max1, y_max2)

    """ Create meshgrid of x and y axis"""
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.05), np.arange(y_min, y_max, 0.05))
    Z1, Z2 = calculate_posterori(densEst1, densEst2, xx, yy)
    Z = decision(Z1, Z2)        #make a decision of the probability

    """ create decision boundary with both dataset in one plot"""
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
    plt.title('p(x|C): red: decision for densEst1, Blue: decision for densEst2')
    # Plot also the training points
    plt.scatter(densEst1.x, densEst1.y, marker='x', color='red', label='densEst1')
    plt.scatter(densEst2.x, densEst2.y, marker='x', color='blue', label='densEst2')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.legend()
    plt.show()

def main():
    #inizialization
    x_densEst1, y_densEst1 = get_data('dataSets/densEst1.txt')
    x_densEst2, y_densEst2 = get_data('dataSets/densEst2.txt')

    densEst1 = Data(x_densEst1, y_densEst1, 'densEst1')
    densEst2 = Data(x_densEst2, y_densEst2, 'densEst2')
```

```
    """ 2d) Class Density """
    show_data(densEst1, densEst2)
    multi_gaussian(densEst1)
    multi_gaussian(densEst2)

    """ 2e) Posterior """
    multi_single_plot(densEst1, densEst2)

if __name__ == '__main__':
    main()
```

2.6 Bayesian Estimation

- State the generic case of Bayesian linear regression with data $< \vec{X}, \vec{Y} >$ and parameter θ:
  - Separating two Classes form each other with a regression line. The aim is not not to find the single best value of the model parameters, but rather to determine the posterior distribution for the model parameters. Also, we want to maximize the Likelihood. This maximization of the likelihood can lead to excessively complex model and over-fitting. With Bayesian treatment of linear regression it avoid the over-fitting problem of the maximum likelihood and also determining the model complexity using the trainingdata alone.
- What do we assume about the data, the model and the parameter:
  - Data: $\vec{X}, \vec{Y} \in \mathbb{R}^{1\times N}$ and they are independent
  - Model: Supervised Learning. The Label of the data are given.
  - Parameter θ: Is a normal distribution, with $\theta = (\mu, \sigma^2)$
- Formulate the posterior distribution for your model parameters given the data, i.e., p($\theta \mid \vec{X}, \vec{Y}$), and derive its mean and covariance, assuming that the model of the output variable is a Gaussian distribution with a fixed variance.

  Let $\theta = (\mu, \sigma^2)$ The posterior distribuiton can be written as:

$$p(\theta \mid X_1, \ldots, X_n) = \frac{p(X_1, \ldots, X_n \mid \theta)p(\theta)}{p(X_1, \ldots, X_n)} = \frac{\mathcal{L}_n(\theta)p(\theta)}{p(X_1, \ldots, X_n)} \propto \mathcal{L}_n(\theta)p(\theta)$$

  where $\mathcal{L}_n(\theta) = \prod_{i=1}^{n} p(X_i \mid \theta)$ is the likelihood funciton and

$$p(X_1, \ldots, X_n) = \int p(X_1, \ldots, X_n)p(\theta)d\theta = \int L_n(\theta)p(\theta)d\theta$$

  is the normalizing constant (evidence).

  We also know, if we estimate the distribution of the mean:

$$p(\mu \mid X) = \frac{p(X \mid \mu)p(\mu)}{p(X)}$$

  with the prior:

$$p(\mu) \sim \mathcal{N}(\mu_0, \sigma_0^2) \Rightarrow p(\mu) = \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp(\frac{-(\mu - \mu_0)^2}{2\sigma_0^2})$$

$$p(X \mid \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(\frac{-(x - \mu)^2}{2\sigma^2})$$

  With n independent observation X = $(X_1, X_2, \ldots, X_n)$, such that:

$$p(\mu \mid x) \propto p(x \mid \mu)p(\mu) = p(\mu)p(x_1 \mid \mu)p(x_2 \mid \mu) \ldots p(x_n \mid \mu)$$

and the Formula from above:

$$
\begin{aligned}
p(\mu \mid x) &= \frac{p(x \mid \mu)p(\mu)}{\int p(x \mid \mu)p(\mu)d\mu} = \frac{p(x \mid \mu)p(\mu)}{p(x)} \propto p(x \mid \mu)p(\mu) \\
&= \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp(\frac{-(\mu-\mu_0)^2}{2\sigma_0^2}) * \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp(\frac{-(x_i-\mu)^2}{2\sigma^2}) \\
&= \frac{1}{(2\pi)^{\frac{n+1}{2}}\sqrt{\sigma_0^2\sigma^{2n}}} \exp(\frac{-\mu^2+2\mu\mu_0-\mu_0^2}{2\sigma_0^2} - \sum_{i=1}^{n}\frac{x_i^2-2\mu x_i+\mu^2}{2\sigma^2}) \\
&= \text{const} * \exp(\frac{-\mu^2(\sigma^2+n\sigma_0^2)+2\mu(\mu_0\sigma^2+\sigma_0^2x_1+\cdots+\sigma_0^2x_n)-(\mu_0^2\sigma^2+\sigma_0^2x_1^2+\cdots+\sigma_0^2x_n^2)}{2\sigma_0^2\sigma^2}) \\
&\propto \exp(\frac{-\mu^2+2\mu\frac{\mu_0\sigma^2+\sum_{i=1}^{n}\sigma_0^2x_i}{\sigma^2+n\sigma_0^2} - (\frac{\mu_0\sigma^2+\sum_{i=1}^{n}\sigma_0^2x_i}{\sigma^2+n\sigma_0^2})^2}{2\frac{\sigma_0^2\sigma^2}{\sigma^2+n\sigma_0^2}}) * \exp(-\frac{\mu_0^2\sigma^2+\sum_{i=1}^{n}\sigma_0^2x_i^2}{2\sigma_0^2\sigma^2}) \\
&\propto \exp(-\frac{(\mu-\frac{\mu_0^2\sigma^2+\sum_{i=1}^{n}\sigma_0^2x_i^2}{\sigma^2+n\sigma_0^2})^2}{2\frac{\sigma_0^2\sigma^2}{\sigma^2+n\sigma_0^2}})
\end{aligned}
$$

Letting:

$$
\begin{aligned}
\sigma_1 &= \frac{\sigma_0^2\sigma^2}{\sigma^2+n\sigma_0^2} = \frac{1}{\sigma_0^{-2}+n\sigma^{-2}} \\
\mu_1 &= \frac{\mu_0^2\sigma^2+\sum_{i=1}^{n}\sigma_0^2x_i^2}{\sigma^2+n\sigma_0^2} = \frac{\mu_0\sigma_0^{-2}+\sum_{i=1}^{n}x_i\sigma^{-2}}{\sigma_0^{-2}+n\sigma^{-2}} = \sigma_1^2(\mu_0\sigma_0^{-2}+\sum_{i=1}^{n}x_i\sigma^{-2})
\end{aligned}
$$

Now we can write it as:

$$
\begin{aligned}
\sigma_1^2 &= (\frac{1}{\sigma_0^2}+\frac{n}{\sigma^2})^{-1} \\
\mu_1 &= \sigma_1^2(\frac{\mu_0}{\sigma_0^2}+\frac{\overline{x}n}{\sigma^2})
\end{aligned}
$$

We can alternatively write these formulas as:

$$
\begin{aligned}
\mu_n &= \frac{N\sigma_0\overline{x}+\sigma^2\mu}{N\sigma_0^2+\sigma^2} \\
\frac{1}{\sigma_N^2} &= \frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}
\end{aligned}
$$

Multidimensional case:

$$
p\left(\boldsymbol{\theta}|\boldsymbol{x},y\right) \propto p\left(y|\boldsymbol{x},\boldsymbol{\theta}\right)p\left(\boldsymbol{\theta}\right).
$$

Assuming multivariate Gaussian prior for $p\left(\boldsymbol{\theta}\right)$,

$$
p\left(\boldsymbol{\theta}\right) = p\left(\boldsymbol{\theta}|\boldsymbol{\mu}_\theta,\boldsymbol{\Sigma}_\theta\right) \sim \mathcal{N}_p\left(\boldsymbol{\theta}|\boldsymbol{\mu}_\theta,\boldsymbol{\Sigma}_\theta\right).
$$

Assuming the output is Gaussian distribution with a fixed variance, $\hat{y} \sim \mathcal{N}\left(\hat{y}|\mu_y,\sigma_y^2\right) = \mathcal{N}\left(\hat{y}|\boldsymbol{\theta}^T\boldsymbol{x},\sigma_y^2\right)$.
Given N samples $\{\boldsymbol{x}_n,y_n\}_{n=1}^N$ contained in output vector $\boldsymbol{y} \in \mathbb{R}^N$ and design matrix $\boldsymbol{X} \in \mathbb{R}^{N\times p}$, the likelihood is

$$
p\left(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}\right) \sim \mathcal{N}_N\left(\boldsymbol{y}|\boldsymbol{X}\boldsymbol{\theta},\sigma_y^2\boldsymbol{I}\right).
$$

The posterior is then a multiplication of two multivariate Gaussians,

$$
\begin{aligned}
p\left(\boldsymbol{\theta}|\boldsymbol{x},y\right) &\propto p\left(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}\right)p\left(\boldsymbol{\theta}\right) \\
&\propto \mathcal{N}_N\left(\boldsymbol{y}|\boldsymbol{X}\boldsymbol{\theta},\sigma_y^2\boldsymbol{I}\right)\mathcal{N}_p\left(\boldsymbol{\theta}|\boldsymbol{\mu}_\theta,\boldsymbol{\Sigma}_\theta\right) \\
&\propto c\cdot\mathcal{N}\left(\cdot|\boldsymbol{\mu}_{\text{post}},\boldsymbol{\Sigma}_{\text{post}}\right),
\end{aligned}
$$

where the posterior mean

$$\boldsymbol{\mu}_{\text{post}} = \frac{\left(\left(\sigma_y^2 \boldsymbol{I}\right)^{-1} \boldsymbol{X}\boldsymbol{\theta} + \boldsymbol{\Sigma}_\theta^{-1}\boldsymbol{\mu}_\theta\right)}{\left(\left(\sigma_y^2 \boldsymbol{I}\right)^{-1} + \boldsymbol{\Sigma}_\theta^{-1}\right)^{-1}},$$

the posterior covariance

$$\boldsymbol{\Sigma}_{\text{post}} = \left(\left(\sigma_y^2 \boldsymbol{I}\right)^{-1} + \boldsymbol{\Sigma}_\theta^{-1}\right)^{-1}.$$

with $\boldsymbol{I}$ identity matrix in appropriate dimension [1].

- What do we do when we want to predict a new point?
  - We want the quality of the prediction. That means, we want to minimize the error of the calculation (i.e. RSS (Residual sum of squares), MSE (Mean squared error), etc.). And that means, we maximize the Likelihood, i.e to approximate the probability of posterior. The Error minimization RSS corresponds to maximum likelihood if normal distribution is given (regression)
- Which are the advantages of being Bayesian?
  - By Bayesian regression, we introduce a prior to the parameter θ, $p(\theta)$, that may help against overfitting and instabilities
  - Bayesian helps us to use prios to help us with normalization.
  - If we have new information available, the previous posterior distribution can be used as a prior
  - it provides a wide range of models (hierarchical models and missing data problems)[2]

Tutor solution:

- Assumptions:
  - the data is identically independently distributed (i.i.d),
  - our model is linear in the parameters with some additional Gaussian noise, i.e. $y = f(\vec{x}, \vec{\theta}) + \epsilon$, where $\vec{\theta}$ are parameters we want to learn and $\epsilon \sim \mathcal{N}(0, \sigma_y^2)$
  - prior distribution over the parameters $p(\vec{\theta} \mid \vec{\mu}_{\vec{\theta}}, \vec{\Sigma}_{\vec{\theta}})$
- Given our second assumption, we can refurmulate the model as Given our third assumption, we get the posterior distribution for $\vec{\theta}$ by using Bayes'theorem

$$p(\vec{\theta} \mid \vec{X}, \vec{Y}) = \frac{p(\vec{Y} \mid \vec{X}, \vec{\theta}, \sigma_y^2) p(\vec{\theta} \mid \vec{\mu}_{\vec{\theta}}, \vec{\Sigma}_{\vec{\theta}})}{\vec{Y}}$$

$$\Downarrow$$

$$p(\vec{\theta} \mid \vec{X}, \vec{Y}) \propto p(\vec{Y} \mid \vec{X}, \vec{\theta}, \sigma_y^2) p(\vec{\theta} \mid \vec{\mu}_{\vec{\theta}}, \vec{\Sigma}_{\vec{\theta}}) = \mathcal{N}\vec{\mu}\vec{\Sigma}$$

- As everythin is Gaussian, we can comput the posterior in closed form (see Bishop, 2006, Eq. 1.68, 1.70, 2.115)(as stated in the assignment, we assume that σ_y^2 is constant)

$$\vec{\Sigma}^{-1} = \vec{\Sigma}_{\vec{\theta}}^{-1} + \frac{1}{\sigma_y^2}\vec{X}^T\vec{X}$$

$$\vec{\mu} = \vec{\Sigma}(\vec{\Sigma}_{\vec{\theta}}^{-1}\vec{\mu}_{\vec{\theta}} + \frac{1}{\sigma_y^2}\vec{X}^T\vec{X})$$

- When we want to predict a new point, we have to marginalize the model across the uncertainty in the parameters (we omit the dependencies on the means and covariances for simplicitys sake)

$$p(y \mid \vec{x}, \vec{Y}) = \int p(y \mid \vec{x}, \vec{\theta}) p(\vec{\theta} \mid \vec{X}, \vec{Y}) d\vec{\theta}$$

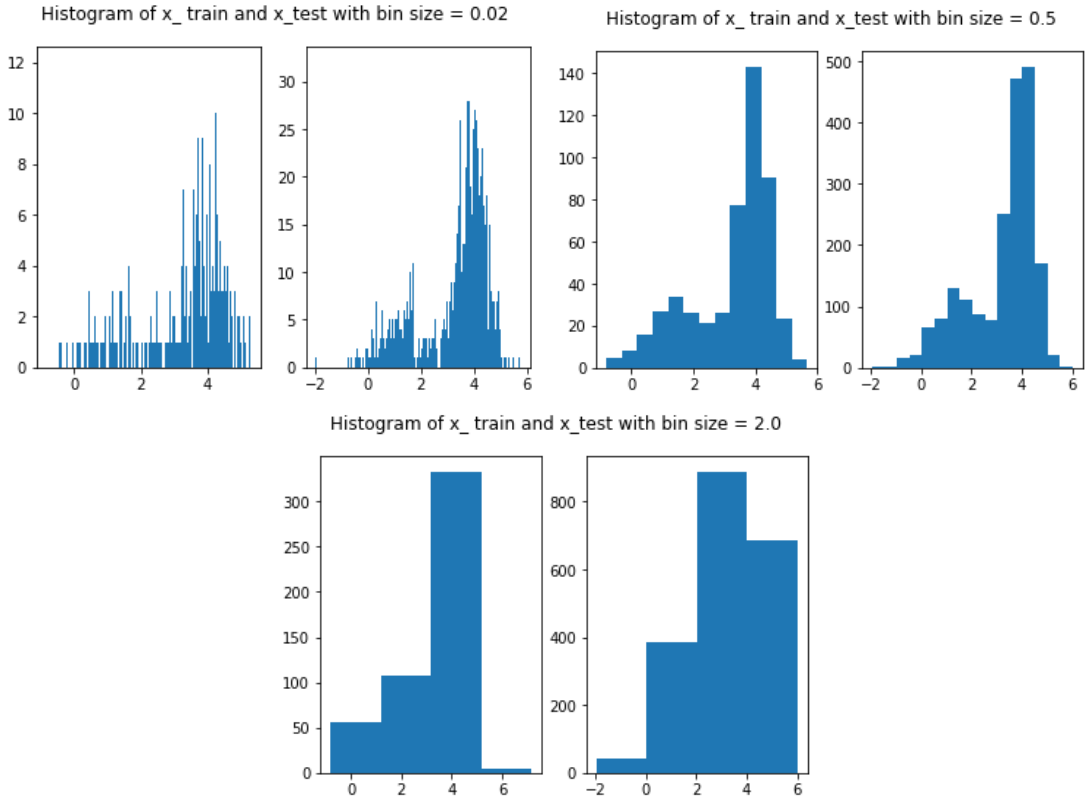
- The main advantage of this approach is that we do not have tho fix the parameters of the model. instead of using one set of parameters, we average over all of the possible ones. Additionally, we are able to introduce a prior probability over the parameters. As a result we get a probability indecating the uncertainty of the model predictions. (more information in tutors exercise hour)

[1] http://compbio.fmph.uniba.sk/vyuka/ml/old/2008/handouts/matrix-cookbook.pdf
[2] Source: https://www.cpp.edu/~djmoriarty/wed/bayes_handout.pdf

3 Non-parametric Density Estimation

3.1 Histogram

The following figures show the histograms of the train and test data using different bin sizes (widths).



Intuitively, it is better to use the bin size of 0.5, because we can observe that using bin width that is too small (0.02) and seams to be over-fitting the data, the density is not smooth enough, while with bin width that is too big (2), the density is too smooth.

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

def get_counts(x, binsize):
    xmin = min(x)
    xmax = max(x)
    which_bin = np.floor((x-xmin)/binsize)
    edges = np.arange(xmin, xmax + binsize, binsize)
    Nbins = edges.shape[0]-1
    counts = np.zeros(Nbins)
    for n in range(Nbins):
        counts[n] = np.count_nonzero(which_bin==n)
    return (edges[1:]-binsize/2), counts

def plot_my_hist(x, y, binsize, title): # To plot two distributions side by side
    fig, axs = plt.subplots(1,2)
    fig.suptitle(title)

    midpts, counts = get_counts(x, binsize)
    axs[0].bar(x=midpts, height=counts, width=binsize)

    midpts, counts = get_counts(y, binsize)
    axs[1].bar(x=midpts, height=counts, width=binsize)

    plt.show()

def main():
    x_train = np.genfromtxt("nonParamTrain.txt")
    x_test = np.genfromtxt("nonParamTest.txt")

```

```
    for b in [0.02, 0.5, 2.0]:
        plot_my_hist(x_train, x_test, b, "Histogram of x_ train and x_test with bin size = "+ str(b))

if __name__ == '__main__':
    main()
```

3.2 Kernel Density Estimate

Gaussian kernel in 1 dimension

$$k(u) = \frac{1}{\sqrt{2\pi}} \exp\left\{-(u)^2\right\},$$

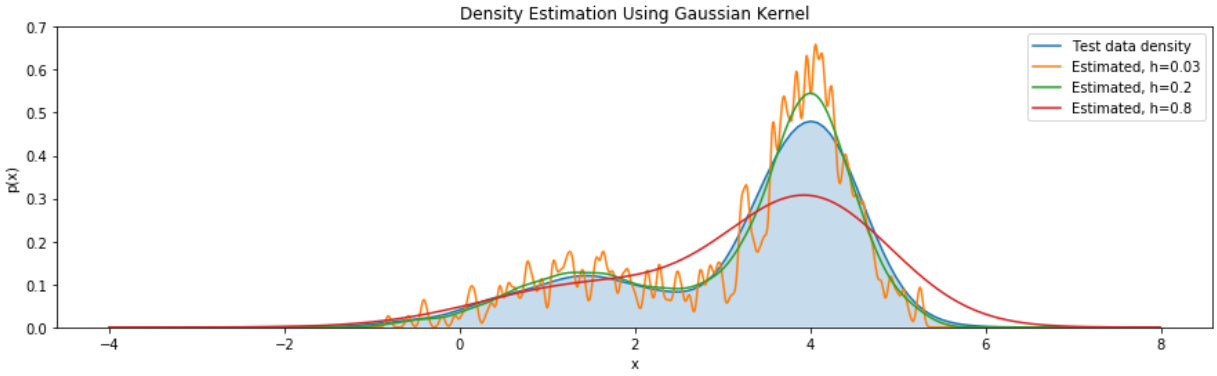
with

$$K(x;h) = \sum_{n=1}^{N} k\left(\frac{x - x^{(n)}}{h}\right)$$

where N is the number of total points and K is the number of points falling in the region R. Then, the density estimate (parameterized with h): is

$$p(x;h) \approx \frac{K(x;h)}{Nh} = \frac{1}{Nh} \sum_{n=1}^{N} k\left(\frac{x - x^{(n)}}{h}\right).$$

The estimated density was calculated using the training data with $h = 0.03, 0.2, 0.8$. The test data density was plotted using the kdeplot function from Seaborn. We can observe that for very small bandwidth (h=0.03), the curve is too grainy and the difference (error) w.r.t to the test data is relatively large. For h=0.8, the curve is overly smoothed and again the error w.r.t test data is large. In this case the best is h=0.2, where we can well distinguish the two modes (two peaks) and the closest one to the test data.



The likelihood function with a set of i.i.d data $x^{(1)}, \ldots, x^{(M)}$ is

$$\mathcal{L} = \prod_{m=1}^{M} p\left(x^{(m)};h\right) = \frac{1}{(Nh)^M} \prod_{m=1}^{M} K\left(x^{(m)};h\right).$$

And the log likelihood

$$\ell = \ln \mathcal{L} = \sum_{m=1}^{M} \ln p\left(x^{(m)};h\right) = -M \ln(Nh) + \sum_{m=1}^{M} \ln K\left(x^{(m)};h\right).$$

Log likelihood:

- with h=0.03, training $\ell \approx -674.73$, test $\ell \approx -2812.11$
- with h=0.2, training $\ell \approx -717.02$, test $\ell \approx -2877.33$
- with h=0.8, training $\ell \approx -795.66$, test $\ell \approx -3192.49$

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

def H_gauss(u): #1-D Gaussian kernel
    return np.exp(-(u)**2/2) / np.sqrt(2*np.pi)

```

```
def K_gauss(X, data, h):
    M = X.shape[0]
    N = data.shape[0]
    K = np.zeros(M)
    for m in range(M):
        K_sum = 0
        for n in range(N):
            K_sum += H_gauss((X[m]-data[n])/h)
        K[m] = K_sum
    return K

def LL(p):
    return np.sum(np.log(p))

def main():
    x_train = np.genfromtxt("nonParamTrain.txt")
    x_test = np.genfromtxt("nonParamTest.txt")
    Ntrain = x_train.shape[0]
    Ntest = x_test.shape[0]
    X = np.arange(-4,8,0.01)

    ll_train = np.zeros(3)
    ll_test = np.zeros(3)
    i = 0

    plt.figure(figsize=(15,4))
    sns.kdeplot(x_test, shade=True, label='Test data density')
    for h in [0.03, 0.2, 0.8]:
        p_train = K_gauss(x_train, x_train, h) / Ntrain / h
        p_test = K_gauss(x_test, x_test, h) / Ntest / h
        p_X_train = K_gauss(X, x_train, h) / Ntrain / h

        # Log likelihood
        ll_train[i] = LL(p_train)
        print('Log likelihood of the training data with h='+str(h)+' is '+str(ll_train[i]))
        ll_test[i] = LL(p_test)
        print('Log likelihood of the test data with h='+str(h)+' is '+str(ll_test[i]))
        i += 1

        # Plotting
        plt.plot(X, p_X_train, label='Estimated, h='+str(h))
    plt.ylim((0.0,0.7))
    plt.legend()
    plt.title('Density Estimation Using Gaussian Kernel')
    plt.ylabel('p(x)')
    plt.xlabel('x')
    plt.show()

if __name__ == '__main__':
    main()
```

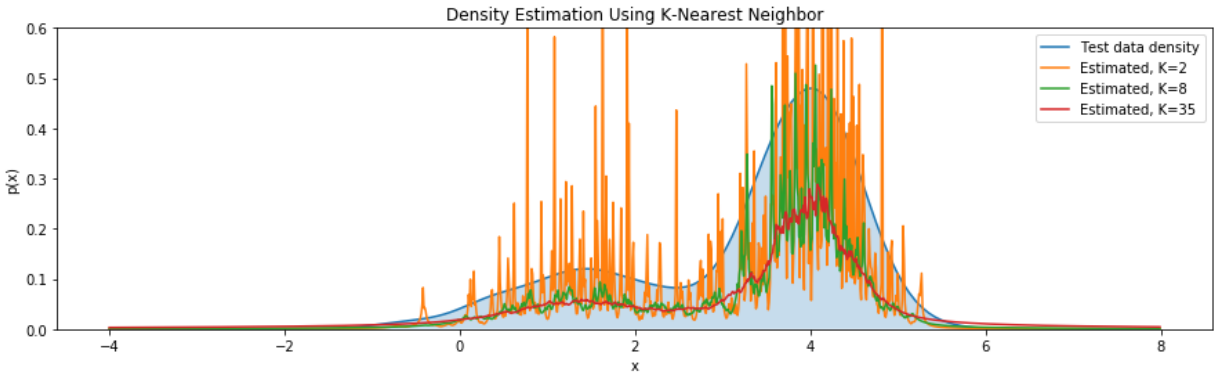
3.3 K-Nearest Neighbours

In the KNN method we grow the volume surrounding the estimation point x so that it encloses a total of K points. The density estimate then becomes

$$p\left(x; K\right) \approx \frac{K}{NV},$$

where for 1-D, $V = 2R_K$, where R_K is the distance to its K-th closest neighbor [3].

[3] http://faculty.washington.edu/yenchic/18W_425/Lec7_knn_basis.pdf



The estimated density was calculated using the training data. The test data density was plotted using the kdeplot function from Seaborn. We can observe in the image that using $K = 2$, the estimate gives very bad result. With $K = 8$ is somewhat closer to the test density but still rather granular, not smooth enough. In this case, in our opinion, the best case is 35 where the result is smooth enough.

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

def get_furthestKNNidx(x, data, K):
    distances = np.abs(data-x)
    return np.argsort(distances)[K-1] #index of furthest K nearest neighbor

def get_density(X, data, K):
    NX = X.shape[0]
    p_X = np.zeros(NX)
    for n in range(NX):
        furthest_knn_idx = get_furthestKNNidx(X[n], data, K) #index of furthest K nearest neighbor
        R = np.abs(data[furthest_knn_idx] - X[n])#max distance
        p_X[n] = K/NX/2/R
    return p_X

def main():
    x_train = np.sort(np.genfromtxt("nonParamTrain.txt"))
    x_test = np.sort(np.genfromtxt("nonParamTest.txt"))
    X = np.arange(-4,8,0.01)

    # Plotting
    plt.figure(figsize=(15,4))
    sns.kdeplot(x_test, shade=True, label='Test data density')
    for K in [2,8,35]:
        p_X_train = get_density(X, x_train, K)
        plt.plot(X, p_X_train,label='Estimated, K='+str(K))
    plt.ylim((0.0,0.6))
    plt.legend()
    plt.title('Density Estimation Using K-Nearest Neighbor')
    plt.ylabel('p(x)')
    plt.xlabel('x')
    plt.show()

if __name__ == '__main__':
    main()
```

3.4 Comparision of the Non-Parametric Methods

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?

If we use only the training data for evaluating the performance of different parameters, then we are prone to overfit to the data. That means that we can explain the training data very well with our model, but we loose the generalization capability. As a result, when we evaluate our model on the testing data the performance degrades. Note that the testing data have been generate by the same process used for the training data!

The KDE estimator log-likelihoods on the testing data are [-inf, -2904,342, -3188.8334]. The KNN estimator log-likelihoods are [-2298.840, -2708,403, -2786.484]. We observe that:

- We get the best performance using KNN estimator with K=2
- The log-likelihoods are considerably smaller than the ones on the training data
- However, from the plot we see that KNN with K=2 gives us a very noisy and overfitting density. The same for K=8
- With K=35 we get a good density, and its log-likelihood is higher than KDE with $\sigma = 0.2$, Therefore, KNN with K=35 would be a good choice

But why is the log-likelihood not truly a good indicator in this case?

- First, KNN does not give you a true distribution, i.e., the integral over the variable domain does not sum up to 1. For instance if we use N=1, we fit a distribution that has extremly high probability at each data point. Ad shown below, this density has even a higher log-likelihood than N=2 but is a terrible choice
- Second, the log-likelihood tells us how well we are fitting the data given some parameter. But it does not tell us how good the parameters are given the data (a.k.a posterior). Therefore, the likelihood is not always a good indicator for model selection. In general, MLE estimators are not always a good choice and sometimes it is better to have a prior over the parameters and use a MAP estimate.
  - THe maximum likelihood estimate (MLE) of a parameter is the value of the parameter that maximizes the likelihood, where the likelihood is a function of the parameter and is actually equal to the probability of the data conditioning on the parameter.
  - Maximum a posteriori (MAP) estimation is the value of the parameter that maximizes hte entire posterior distribution (which is calculated using the likelihood). A MAP estimate is the mode of the posterior distribution.
  - Note that there is no difference between the MLE and MAP estimate if the prior distribution we were assuming was a constant

```
plt.figure(figsize=(10,8))
p1_x = knn(x, 1, train_data)
p2_x = knn(x, 8, train_data)
p3_x = knn(x, 35, train_data)
line1, = plt.plot(x,p1_x,linewidth=2.0)
line2, = plt.plot(x,p2_x,linewidth=2.0)
line3, = plt.plot(x,p3_x,linewidth=2.0)
plt.legend([line1, line2, line3], ["K = 1", "K = 8", "K = 35"])
plt.xlabel('x')
plt.ylabel('p_x')

plt.axis([-4, 8, 0, 3])

p1_test = knn(test_data, 1, train_data)
log_like_test_1 = np.sum(np.log(p1_test))
p2_test = knn(test_data, 8, train_data)
log_like_test_2 = np.sum(np.log(p2_test))
p3_test = knn(test_data, 35, train_data)
log_like_test_3 = np.sum(np.log(p3_test))

print('\nlog likelihood on the test set')
print([log_like_test_1, log_like_test_2, log_like_test_3])
```

4 Expectation Maximization

4.1 Gaussian Mixture Update Rules

M-component Gaussian mixture model:

$$\begin{aligned} p\left(\boldsymbol{x}_n|\boldsymbol{\theta}\right) &= \sum_{j=1}^{M} p\left(\boldsymbol{x}_n|j\right) p\left(j\right) \\ &\equiv \sum_{j=1}^{M} \pi_j \mathcal{N}\left(\boldsymbol{x}_n; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right) \end{aligned}$$

where the mixture parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \pi_1, \ldots, \boldsymbol{\mu}_M, \boldsymbol{\Sigma}_M, \pi_M\}$, $\sum_{j=1}^{M} \pi_j = 1$, and in this case $M = 4, \boldsymbol{x}, \boldsymbol{\mu} \in \mathbb{R}^2, \boldsymbol{\Sigma} \in \mathbb{R}^{2\times 2}$.

Given the observed data $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, we want to maximize the log-likelihood function of X w.r.t to the parameters:

$$\arg\max_{\boldsymbol{\theta}} \ell\left(X|\boldsymbol{\theta}\right) = \arg\max_{\boldsymbol{\theta}} \sum_{n=1}^{N} \log \sum_{j=1}^{M} \pi_j \mathcal{N}\left(\boldsymbol{x}_n; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right).$$

E- (Expectation) step: Compute the posterior distribution $p\left(j|\boldsymbol{x}_n\right)$ for each mixture component j and for all data points $n = 1, \ldots, N$. For time step i:

$$\hat{\alpha}_{nj}^{(i)} = \left[\hat{p}\left(j|\boldsymbol{x}_n\right)\right]^{(i)} = \frac{\hat{\pi}_j^{(i-1)} \mathcal{N}\left(\boldsymbol{x}_n; \hat{\boldsymbol{\mu}}_j^{(i-1)}, \hat{\boldsymbol{\Sigma}}_j^{(i-1)}\right)}{\sum_{k=1}^{M} \hat{\pi}_k^{(i-1)} \mathcal{N}\left(\boldsymbol{x}_n; \hat{\boldsymbol{\mu}}_k^{(i-1)}, \hat{\boldsymbol{\Sigma}}_k^{(i-1)}\right)}.$$

M- (Maximization) step: Estimate the parameters using the current values of $\hat{p}\left(j|\boldsymbol{x}_n\right)$. Let $N_j = \sum_{n=1}^{N} \hat{\alpha}_{nj}^{(i)}$, for time step i:

$$\begin{aligned} \hat{\boldsymbol{\mu}}_j^{(i)} &= \frac{1}{N_j} \sum_{n=1}^{N} \hat{\alpha}_{nj}^{(i)} \cdot \boldsymbol{x}_n \\ \hat{\boldsymbol{\Sigma}}_j^{(i)} &= \frac{1}{N_j} \sum_{n=1}^{N} \hat{\alpha}_{nj}^{(i)} \left(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_j^{(i)}\right) \left(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_j^{(i)}\right)^T \\ \hat{\pi}_j^{(i)} &= \frac{N_j}{N}. \end{aligned}$$

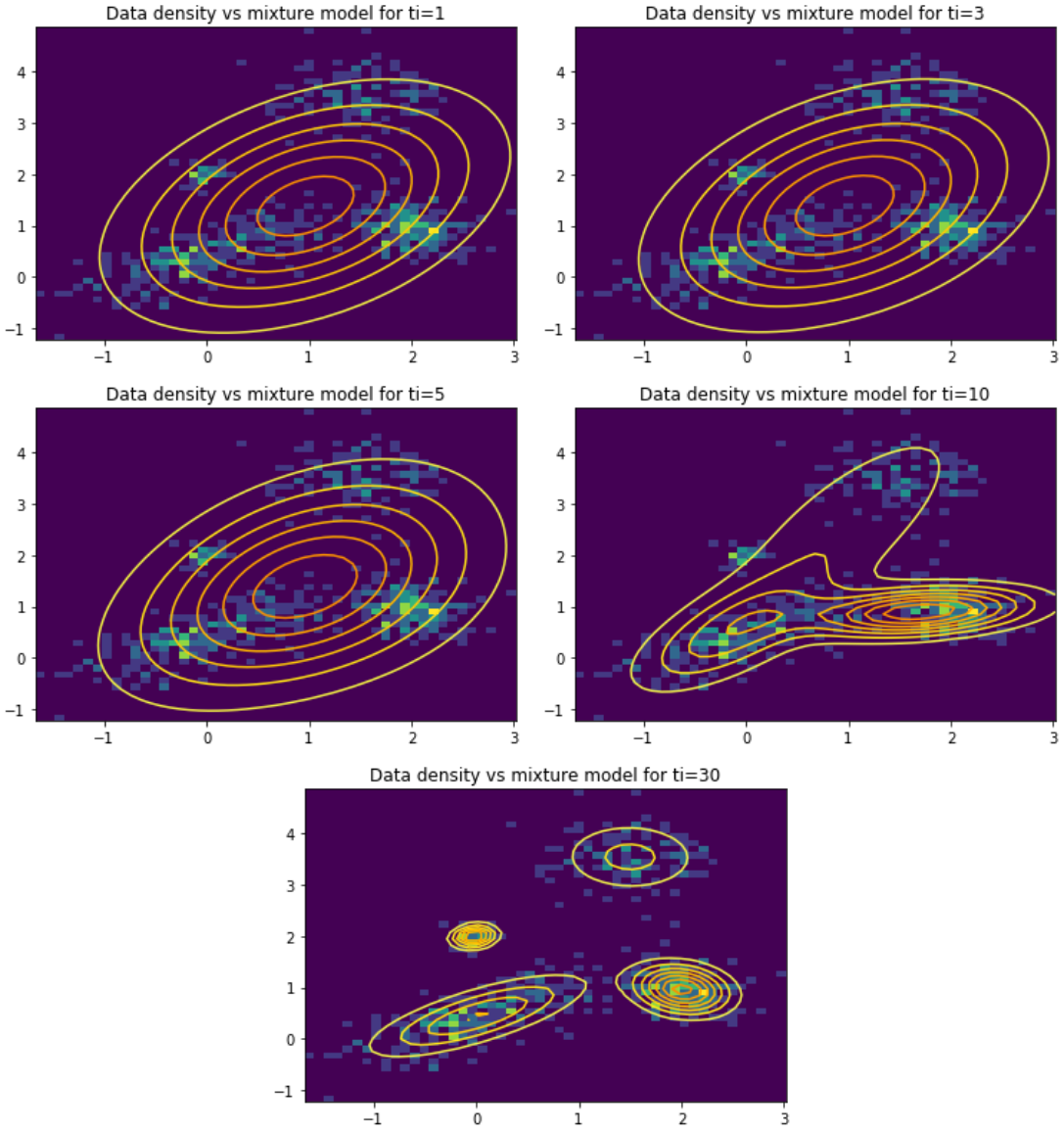
4.2 EM

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

def get_data(name :str) -> np.asarray:
    file = open(name, 'r')
    x1 = []
    x2 = []
    for i in file.readlines():
        line = i.split()
        x1.append(float(line[0]))
        x2.append(float(line[1]))
    file.close()
    return np.row_stack((x1,x2))

def get_cov(X):
    return np.matmul(X,np.transpose(X))/X.shape[1]

def multvar_gauss(x,m,S):
    p = m.shape[0]
    return (2*np.pi)**(-p/2) / np.sqrt(np.linalg.det(S)) * np.exp(-0.5*np.matmul(np.transpose(x-m), np.matmul(np.linalg.
     inv(S),(x-m)) ))

def get_mixpdf(x1,x2,t):
    X=np.array([x1,x2])
    M = int(len(t) / 3)
    pdf = 0
    for j in range(M):
        m = t[j*3]
        S = t[j*3+1]
```

```
        pi = t[j*3+2]
        pdf += pi * multvar_gauss(X,m,S)
    return pdf

def plot_overlay(x,t,title):
    nbins = 50
    data_density = plt.hist2d(x[0,:],x[1,:],bins=nbins)
    x1_edges = data_density[1]
    x2_edges = data_density[2]
    X1,X2 = np.meshgrid(x1_edges, x2_edges)
    mixpdf = np.zeros((nbins+1,nbins+1))
    for i in range(nbins+1):
        for j in range(nbins+1):
            mixpdf[i,j] = get_mixpdf(X1[i,j],X2[i,j],t)
    plt.contour(X1, X2, mixpdf, cmap="Wistia")
    plt.title(title)
    plt.show()

def main():
    #read data
    x = get_data('gmm.txt')
    p = x.shape[0]
    N = x.shape[1]

    #initialization
    M = 4
    pi = 1/M
    all_mu = [None]*M
    all_Sigma = [None]*M
    all_pi = [pi]*M
    np.random.seed=1234
    for j in range(M):
        all_mu[j] = np.random.rand(2)
        all_Sigma[j] = get_cov(x-all_mu[j][:,np.newaxis])

    alpha = np.zeros((M,N))
    steps = 30
    theta = [None]*(steps) #tracking all the values for each step {mu_1, Sigma_1, pi_1, ...}
    L = np.zeros(steps) #log likelihood

    for i in range(steps):
        t = [None] * (3*M)

        #E-Step
        for n in range(N):
            alpha_denom = 0
            beta = np.empty(M)
            for j in range(M):
                bn = multvar_gauss(x[:,n],all_mu[j],all_Sigma[j])
                beta[j] = all_pi[j]*bn
                alpha_denom += beta[j]

            alpha[:,n] = beta/alpha_denom

        #M-step
        Nj = np.sum(alpha, axis=1)
        for j in range(M):
            all_mu[j] = np.sum(alpha[j,:]*x, axis=1) / Nj[j]
            s = np.zeros((p,p))
            for n in range(N):
                X = x[:,n]-all_mu[j]
                X = X[:,np.newaxis]
                s = s + alpha[j,n]*np.matmul(X,np.transpose(X))
            all_Sigma[j] = s / Nj[j]
        all_pi = Nj / N

        t[0:3*M:3] = all_mu
        t[1:3*M:3] = all_Sigma
        t[2:3*M:3] = all_pi
        theta[i] = t

        #log likelihood
        ll = 0
        for n in range(N):
            l = 0
            for j in range(M):
                l += all_pi[j] * multvar_gauss(x[:,n],all_mu[j],all_Sigma[j])
            ll += np.log(l)
```

```
        L[i] = ll

    for i in [1,3,5,10,30]:
        plot_overlay(x,theta[i-1], 'Data density vs mixture model for ti='+ str(i))

    # plot log likelihood
    plt.plot(L)
    plt.ylabel("Log likelihood")
    plt.xlabel("# iterations")
    plt.title("Log likelihood updates")
    plt.show()

    # plot mixture scales
    for j in range(M):
        pi = [x[j*3+2] for x in theta]
        plt.plot(np.arange(1,31), pi, label='pi'+ str(j+1))
    plt.ylabel("pi")
    plt.xlabel("# iterations")
    plt.title("Updates of mixture scaling parameter")
    plt.legend()
    plt.show()

if __name__ == '__main__':
    main()
```

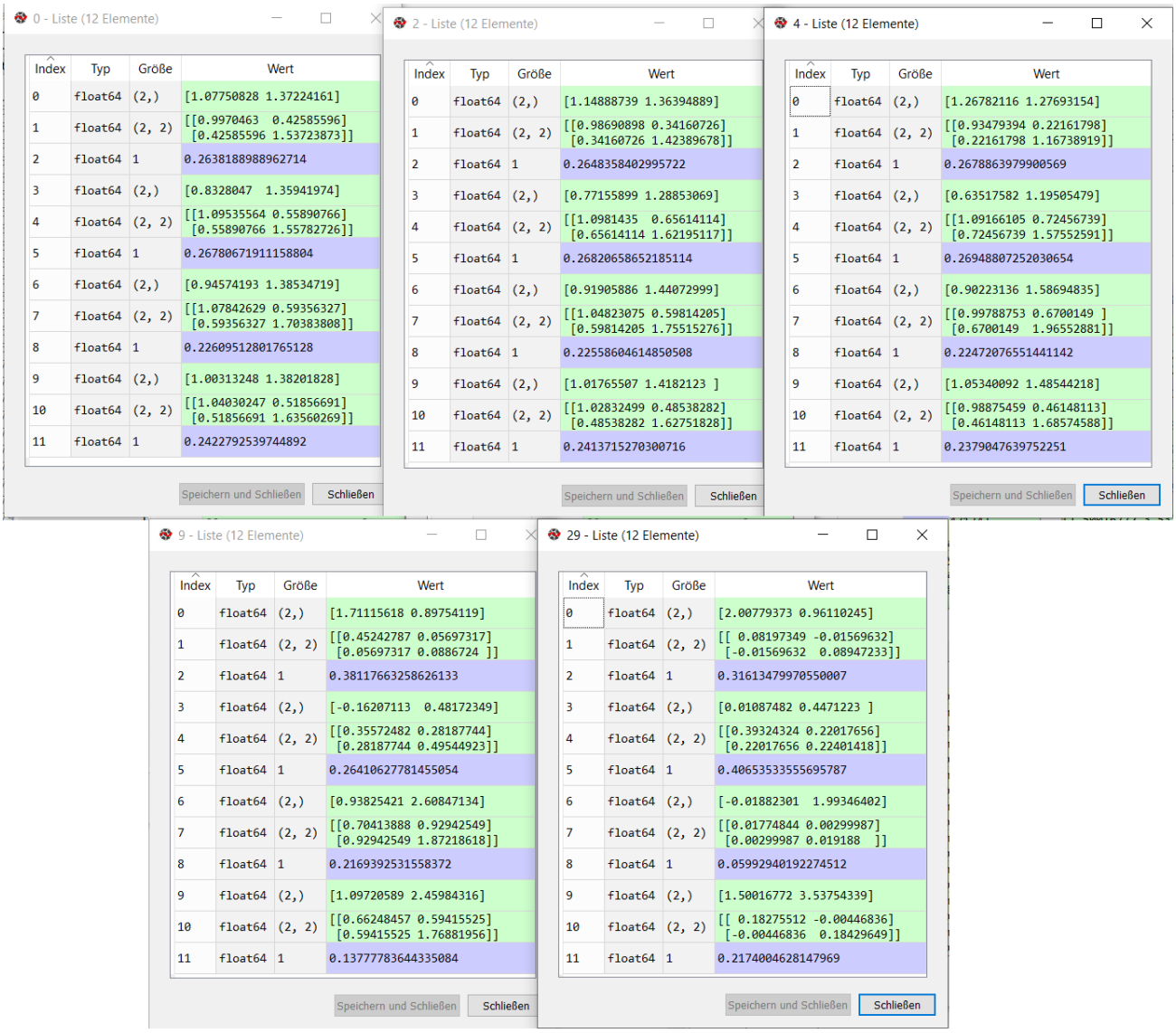
Below we see the density of the observed data overlayed with the evolution of the mixture models at iterations $t_i = 1, 3, 5, 10, 30$.



The figure below shows the parameter updates for $t_i = 1, 3, 5, 10, 30$, where:

- indices 0,3,6,9 are $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3, \boldsymbol{\mu}_4$
- indices 1,4,7,10 are $\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \boldsymbol{\Sigma}_3, \boldsymbol{\Sigma}_4$
- indices 2,5,8,11 are $\pi_1, \pi_2, \pi_3, \pi_4$

After $t_i = 30$, $\pi_1, \pi_2, \pi_3, \pi_4$ are pretty stable at ≈ 0.31, 0.41, 0.06, and 0.22, respectively.



The next figures show the evolution of the the mixture scaling parameters π_j and the log likelihood function.