# IFT6390 - Homework_1

Julie Kienzle & Jinfang Luo

September 28, 2018

## 1 Small exercise on probabilities [10 points]

### Problem

A few years ago, a study was carried out with doctors in the United States, in order to measure their "probabilistic intuition". It included the following question: A percentage of 1.5% of women in their 40s who take a routine test (mammogram) have breast cancer. Among women that have breast cancer, there is a 87% chance that the test is positive. In women that do not have breast cancer, there is a probability of 9.6% that the test is positive.
A woman in her forties who has passed this routine test receives a positive test result. What is the probability that it is actually breast cancer?
A) more than 90%
B) between 70% and 90%
C) between 50% and 70%
D) between 30% and 50%
E) between 10% and 30%
F) less than 10%
95% of doctors surveyed responded B). What do you think? Formalize the question and calculate the exact probability. Hint: use Bayes rule ...

#### Solution

In order to formalize the question, we first need to define two events:
$A$: the event of having breast cancer.
$B$: the event that the test receives a positive result.

The question can now be written mathematically. We want to find $P(A|B)$: the probability that the women has breast cancer knowing that the test gave a positive result.

We can withdraw some information from the problem's description:

- P($A$) = 1.5% (probability of having breast cancer).
- P($A^C$) = 100%-1.5% = 98.5% (probability of not having breast cancer).
- P($B|A$)= 87% (probability that the test in positive knowing that the person has breast cancer).
- P($B|A^C$)= 9.6% (probability that the test in positive knowing that the person does not have breast cancer).

According to Bayes rule, the probability that we want to calculate can be obtained by this following equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

We have all the information needed except for $P(B)$ that we don't know. But $P(B)$ can be written as:

$$P(B) = P(B|A)P(A) + P(B|A^C)P(A^C)$$

So:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|A^C)P(A^C)} = \frac{0.87 * 0.015}{0.87 * 0.015 + 0.096 * 0.985} = 0.12127 = 12.127\%$$

Doctors should have answered E) between 10% and 30%.This makes sense because 98.5% of the people that take the test do not have breast cancer. And 9.6% of these receive a positive test. Thus, it is much more likely that the test was defective than the person really having breast cancer.

## 2 Curse of dimensionality and geometric intuition in higher dimensions [20 points]

### 2.1 Problem

We consider a hyper-cube in dimension $d$ (this is a generalization of the 2D square and the 3D cube) with side length $c$ (which can be expressed in cm for example). What is the volume $V$ of this hyper-cube?

**Solution**

$$V = c^d (cm^d)$$

### 2.2 Problem

We define a random vector $X$ of dimension $d$ ($x \in \mathbb{R}^d$) distributed uniformly within the hypercube (the probability density $p(x) = 0$ for all $x$ outside the cube). What is the probability density function $p(x)$ for $x$ inside the cube? Indicate which property(ies) of probability densities functions allow you to calculate this result.

**Solution**

The probability density function $p(x)$ for $x$ inside the cube is:

$$p(x) = \frac{1}{c^d}$$

This can be found by taking into account two facts:

- A uniformly distributed variable has a constant probability density function.
- The integral of $p(x)$ on the function's region must be equal to $1$.

We, thus, have to find a constant function that integrates to 1 over the $d$ dimensions's hypercube of side length $c$. Mathematically $p(x) = p(x_1, ..., x_d)$ has to satisfy:

$$\int_0^c ... \int_0^c p(x)dx_1...dx_d = 1 \qquad \text{where } p(x) = \text{constant}$$

$\frac{1}{c^d}$ satisfies that as it can be shown bellow:

$$\int_0^c ... \int_0^c \frac{1}{c^d} dx_1...dx_d = \int_0^c ... \int_0^c \frac{x_1}{c^d}\Big|_0^c dx_2...dx_d = \int_0^c ... \int_0^c \frac{c}{c^d}\Big|_0^c dx_2...dx_d = \frac{c^d}{c^d} = 1$$

## 2.3 Problem

Consider the outer shell (border) of the hypercube of width 3% of $c$ (covering the part of the hypercube extending from the faces of the cube and $0.03c$ inwards). For example, if $c = 100$cm, the border will be 3cm (left, right, top, etc ...) and will delimit this way a second (inner) hypercube of side $100 - 3 - 3 = 94$cm. If we generate a point $x$ according to the previously defined probability distribution (by sampling), what is the probability that it falls in the border area? What is the probability that it falls in the smaller hypercube?

### Solution

The smaller hypercube volume = $(c - 0.03c * 2)^d = (c(1 - 0.06))^d = (0.94c)^d = 0.94^d c^d$ cm$^d$

The probability that it falls in the smaller hypercube = $\frac{\text{Volume of small hypercube}}{\text{Volume of hypercube}} = \frac{94^d c^d}{c^d}$ = $0.94^d$
The probability that it falls in the border area = $1 - 0.94^d$

## 2.4 Problem

Numerically calculate the probability that $x$ will fall in the narrow border for the following values of $d$: 1, 2, 3, 5, 10, 100, 1000.

### Solution

Let's define the following event:
FNB: Falling in the narrow border of the hypercube.

For the following values of $d$: 1, 2, 3, 5, 10, 100, 1000, the probabilities that $x$ will fall in the narrow border are:

For $d = 1$: P(FNB) $= 0.06 = 6\%$
For $d = 2$: P(FNB) $= 0.1164 = 11.64\%$
For $d = 3$: P(FNB) $= 0.1694 = 16.94\%$
For $d = 5$: P(FNB) $= 0.2661 = 26.61\%$
For $d = 10$: P(FNB) $= 0.4614 = 46.14\%$
For $d = 100$: P(FNB) $= 0.9979 = 99.79\%$
For $d = 1000$: P(FNB) $\approx 1 = 100\%$

## 2.5 Problem

What do you conclude about the distribution of points in higher dimensions, which is contrary to our intuition in smaller dimensions?

### 2.5.1 Solution

As the number of dimension increases, the probability that point $x$ falls in the narrow border increases as well. This means that in higher dimensions, there are more points that are very close to the border than in the interior. This is not intuitive because in smaller dimensions it's the contrary.

# 3 Parametric Gaussian density estimation, v.s. Parzen window density estimation [35 points]

In this question we consider a dataset $D = \{x^{(1)}, ..., x^{(n)}\}$ with $x \in \mathbb{R}^d$.

## 3.1 Problem

Suppose we have trained the parameters of an isotropic Gaussian density function on D (by maximizing the likelihood) in order to estimate the probability density function.

(a) Name these parameters and indicate their dimension.
(b) If we learn these parameters using the principle of maximum likelihood estimation, express the formula which will give us the value of the optimal parameters as a function of the data points in $D$. Indicate only the formula that calculates the result, you are not asked to rederive it (the formulas for the maximum likelihood estimator can be found at the end of slide set number 5 on the Gaussian distribution).
(c) What is the algorithmic complexity of this training method, i.e. of the method calculating these parameters?
(d) For a test point $x$, write the function that will give the probability density predicted at point $x$: $\hat{p}_{gauss-isotrop}(x)$=?
(e) What is the algorithmic complexity for calculating this prediction at each new point $x$?

**Solution for a)**

The parameters are:

- The mean $\mu$ of dimension $d \times 1$ (column vector of $d$ dimensions).
- The variance-covariance matrix $\Sigma$ of dimension $d \times d$.

Because we are dealing with an isotropic Gaussian density function, we know that $\Sigma$ is actually a diagonal matrix for which all the non zero entries are of the value $\sigma^2$:

$$\Sigma = \begin{bmatrix} \sigma^2 & & \\ & \ddots & \\ & & \sigma^2 \end{bmatrix}$$

Thus, we could also say that the parameters are in fact:

- The mean $\mu$ of dimension $d \times 1$ (column vector of $d$ dimensions).
- The variance $\sigma^2$ of dimension $1 \times 1$ (scalar).

**Solution for b)**

Let $x^{(t)}$ be the $t^{th}$ point among the dataset containing $n$ points. The formula that will give us the value of the optimal parameters are:

- For the whole vector: $\mu = \frac{1}{n}\sum_{t=1}^{n} x^{(t)}$.
  For every component: $\mu_i = \frac{1}{n}\sum_{t=1}^{n} x_i^{(t)}$ for all $i = 1, ..., d$.
  This was directly taken from the end of slide set number 5.
- $\sigma^2 = \frac{1}{nd}\sum_{t=1}^{n}\sum_{i=1}^{d}(x_i^{(t)} - \mu_i)^2$.
  This can be derived as we did it in question 3.4.d) of the homework. The calculations can be seen bellow. The first equation was obtained by replacing $\sigma_i$ by $\sigma$ for all $i = 1, ..., d$ in the equation (2) for finding $\Sigma$ that can be found in question 3.4.c). For more information on how the first equation was obtained, see equation (3) in question 3.4.c).

Contrarily to the number 3.4.d), we only have to compute the partial derivative $\frac{\partial}{\partial \sigma}$. Indeed, the Gaussian is isotropic and $\sigma = \sigma_1 = ... = \sigma_d$. Thus, we calculate:

$$\begin{aligned}
\frac{\partial}{\partial \sigma}\Big(\frac{1}{n}\sum_{t=1}^{n}\Big(-log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma^d}\Big) + \frac{1}{2}\Big(\frac{(x_1^{(t)} - \mu_1)^2}{\sigma^2} + ... + \frac{(x_d^{(t)} - \mu_d)^2}{\sigma^2}\Big)\Big)\Big) &= 0 \\
\frac{1}{n}\sum_{t=1}^{n}\Big(-\frac{1}{\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma^d}}\Big(-\frac{d}{(2\Pi)^{\frac{d}{2}}\sigma^{d+1}}\Big) - \frac{(x_1^{(t)} - \mu_1)^2}{\sigma^3} - ... - \frac{(x_d^{(t)} - \mu_d)^2}{\sigma^3}\Big) &= 0 \\
\sum_{t=1}^{n}\Big(\frac{(2\Pi)^{\frac{d}{2}}\sigma^d d}{(2\Pi)^{\frac{d}{2}}\sigma^{d+1}} - \frac{(x_1^{(t)} - \mu_1)^2}{\sigma^3} - ... - \frac{(x_d^{(t)} - \mu_d)^2}{\sigma^3}\Big) &= 0 \\
\sum_{t=1}^{n}\Big(\frac{d}{\sigma} - \sum_{i=1}^{d}\frac{(x_i^{(t)} - \mu_i)^2}{\sigma^3}\Big) &= 0 \\
\frac{nd}{\sigma} - \sum_{t=1}^{n}\sum_{i=1}^{d}\frac{(x_i^{(t)} - \mu_i)^2}{\sigma^3} &= 0 \\
\frac{nd}{\sigma} &= \sum_{t=1}^{n}\sum_{i=1}^{d}\frac{(x_i^{(t)} - \mu_i)^2}{\sigma^3} \\
\frac{nd}{\sigma} &= \frac{1}{\sigma^3}\sum_{t=1}^{n}\sum_{i=1}^{d}(x_i^{(t)} - \mu_i)^2 \\
nd\sigma^2 &= \sum_{t=1}^{n}\sum_{i=1}^{d}(x_i^{(t)} - \mu_i)^2 \\
\sigma^2 &= \frac{1}{nd}\sum_{t=1}^{n}\sum_{i=1}^{d}(x_i^{(t)} - \mu_i)^2
\end{aligned}$$

**Solution for c)**

- For $\mu$, the complexity can be easily found by looking at the "For every component" formula written in b). It is of $O(nd)$. Indeed, calculating $\mu_i$ is of complexity $O(n)$ because of the sum from 1 to $n$. We have to do that for every of the $d$ components of the vector $\mu$. This explains the obtained complexity of $O(nd)$.
- For $\sigma^2$, the complexity is of $O(nd)$. Indeed, there are two sums imbricated inside one another. One of the sum goes from 1 to $d$ and the other from 1 to $n$ explaining the obtained complexity of $O(nd)$.

Thus, the total complexity of the training method is of $O(nd)$.

**Solution for d)**

As seen in the slides, the isotropic Gaussian probability density function defined by $\mu$ and $\sigma^2$ is:

$$\hat{p}_{\text{gaussisotrop}}(x) = \frac{1}{(2\Pi)^{\frac{d}{2}}\sigma^d} e^{\frac{-\frac{1}{2}||x-\mu||^2}{\sigma^2}}$$

**Solution for e)**

The complexity for calculating this prediction at each new point $x$ is of $O(d)$. To make it easier to see, let's write our norm this way:

$$||x-\mu||^2 = \sum_{i=1}^{d}(x_i-\mu_i)^2$$

Because of the sum going from 1 to $d$, the complexity for computing the norm is of $O(d)$. The value of that norm is a scalar, so $e$ to that scalar does not increase the complexity. Multiplying it by the scalar $\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma^d}$ does not change anything either. Thus, the complexity is of $O(d)$.

## 3.2 Problem

Now consider that one uses Parzen windows with an isotropic Gaussian kernel of width (standard deviation) $\sigma$ instead, and that these Parzen windows were trained on $D$.
(a) Suppose that the user has fixed $\sigma$. What does the "training/learning" phase of these Parzen windows consist of?
(b) For a test point $x$, write in a single detailed formula (i.e. with exponentials), the function that will give the probability density predicted at point $x$ : $\hat{p}_{Parzen}(x) = ?$
(c) What is the algorithmic complexity for calculating this prediction at each new point $x$?

**Solution for a)**

The "training/learning" phase consists of storing the data. Indeed, $\sigma$ is a hyperparameter that was already fixed by the user. Thus, it does not need to be trained.

**Solution for b)**

As seen in the slides of the course, the probability density for the Parzen window is the following:

$$\hat{p}_{Parzen}(x) = \frac{1}{n}\sum_{t=1}^{n} K(X_t; x) \text{ where } K \text{ is a Kernel}$$

Knowing that our Parzen window uses an isotropic Gaussian Kernel of width $\sigma$. We can write our probability density like this:

$$\hat{p}_{Parzen}(x) = \frac{1}{n}\sum_{t=1}^{n} \frac{1}{(2\Pi)^{\frac{d}{2}}\sigma^d} e^{\frac{-\frac{1}{2}||x-x^{(t)}||^2}{\sigma^2}}$$

**Solution for c)**

The algorithm complexity for calculating this prediction at each new point $x$ is of $O(nd)$. As seen in 3.1.e) calculating a norm is of complexity $O(d)$. The value of that norm is a scalar, so $e$ to that scalar does not increase the complexity. Multiplying it by the scalar $\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma^d}$ does not change anything either. Because of the sum going from 1 to $n$, obtaining the probability density will involve computing $n$ times a term of complexity $O(d)$ explaining the obtained complexity of $O(nd)$.

## 3.3 Problem

Capacity/Expressivity
(a) Which one of these two approaches (parametric Gaussian v.s. Parzen Gaussian kernel) has the highest capacity (in other words, higher expressivity)? Explain.
(b) With which one of these approaches, and in which scenario, are we likely to be over-fitting (i.e. memorizing the noise in our data)?
(c) The value in Parzen windows is usually treated as a hyperparameter, whereas for parametric Gaussian density estimation it is usually treated as a parameter. Why?

**Solution for a)**

Supposing that the user has chosen a good hyperparameter for the Parzen Gaussian approach, we can say that the parametric Gaussian one has the highest capacity. Indeed, it has more parameters making the function more flexible and thus allowing the model to represent more complex functions. The Parzen Gaussian kernel approach does not actually contain any parameters. It has only one hyperparameter that is already fixed.

**Remark:** It is important to say that the choice of the hyperparameter controls the capacity. Thus, if the user has chosen a $\sigma$ that is too small, the Parzen Gaussian approach will be able to represent more complex functions then the Gaussian one.

**Solution for b)**

Supposing again that the user has chosen a good hyperparameter for the Parzen Gaussian approach, we can say that the parametric Gaussian approach is more likely to overfit. Indeed, it has more parameters that must be trained then the Parzen Gaussian kernel one. Two parameters is not a lot. Thus, it takes a certain specific scenario to make this approach overfit. Let us consider a small data set containing several outliers. Training the parameters on such a set will likely make the trained function sensible to the noisy data and will not be able to capture the underground true shape of the distribution. In the case of the Parzen approach, the hyperparameter has already being fixed by the user, thus it is not likely to overfit, if well chosen.

**Remark:** Again, it is important to say that the choice of the hyperparameter controls the capacity. Thus, if the user has chosen a $\sigma$ that is too small, the Parzen Gaussian approach will be very sensible to the noise and will overfit more than the parametric Gaussian one.

**Solution for c)**

The value in Parzen windows is usually treated as a hyperparameter because we are creating a distribution centered on the points that we are examining. We choose the distribution around that point. We choose the weights that we want to give to the neighbors of the point that we are examining. We do not try to find a value that describes well the distribution of the points in the dataset which is the role of the parameters in the parametric Gaussian approach.

## 3.4 Problem

Now consider parametric density estimation with a diagonal Gaussian density function.
(a) Express the equation of a diagonal Gaussian density in $\mathbb{R}^d$. Specify what are its parameters and their dimensions.
(b) Show that the components of a random vector following a diagonal Gaussian distribution are independent random variables.
(c) Using $-log(p(x))$ as the loss, write down the equation corresponding to the empirical risk minimization on the training set $D$ (in order to learn the parameters).
(d) Solve this equation analytically in order to obtain the optimal parameters.

**Solution for a)**

As seen in the slides the equation for the diagonal Gaussian density in $\mathbb{R}^d$ is:

$$p(x) = \frac{1}{(2\Pi)^{\frac{d}{2}}\sqrt{|\Sigma|}}e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

where $\Sigma$ is a diagonal variance-covariance matrix as can be seen bellow:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_d^2 \end{bmatrix}$$

Thus, the parameters are:

- The mean $\mu$ of dimension $d \times 1$ (column vector of $d$ dimensions).
- The variance-covariance matrix $\Sigma$ of dimension $d \times d$.

**Solution of b)**

To show that the components of a random vector following a diagonal Gaussian distribution are independent, we need to show that:

$$p(x) = p(x_1, ..., x_d) = p(x_1) * ... * p(x_d)$$

The probability density function of $x_i$ will follow a univariate Gaussian distribution of mean $\mu_i$ and variance $\sigma_i^2$. Its density can, thus, be written as the following:

$$p(x_i) = \frac{1}{(2\Pi)^{\frac{1}{2}}\sigma_i} e^{-\frac{1}{2}\frac{(x_i-\mu_i)^2}{\sigma_i^2}}$$

If the components of the random vector are really independent, then, we would be able to write our diagonal Gaussian probability function as $\prod_{i=1}^{d} p(x_i)$.

Let's show that it is possible:

$$p(x_1, .., x_d) = \frac{1}{(2\Pi)^{\frac{d}{2}}\sqrt{|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

Knowing that the inverse of a diagonal matrix is a diagonal matrix where every element on the diagonal is its reciprocal, the exponent can be written as the following:

$$\begin{aligned} -\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu) &= -\frac{1}{2}[x_1-\mu_1, ...x_d-\mu_d] \begin{bmatrix} \frac{1}{\sigma_1^2} & & \\ & \ddots & \\ & & \frac{1}{\sigma_d^2} \end{bmatrix} \begin{bmatrix} x_1-\mu_1 \\ \vdots \\ x_d-\mu_d \end{bmatrix} \\ &= -\frac{1}{2}\Big(\frac{(x_1-\mu_1)^2}{\sigma_1^2} + ... + \frac{(x_d-\mu_d)^2}{\sigma_d^2}\Big) \end{aligned} \quad (1)$$

So

$$\begin{aligned} p(x_1, .., x_d) &= \frac{1}{(2\Pi)^{\frac{d}{2}}\sqrt{|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)} \\ &= \frac{1}{(2\Pi)^{\frac{d}{2}}\sqrt{|\Sigma|}} e^{-\frac{1}{2}\Big(\frac{(x_1-\mu_1)^2}{\sigma_1^2}+...+\frac{(x_d-\mu_d)^2}{\sigma_d^2}\Big)} \\ &= \frac{1}{(2\Pi)^{\frac{d}{2}}\sqrt{|\Sigma|}} e^{-\frac{1}{2}\frac{(x_1-\mu_1)^2}{\sigma_1^2}} * ... * e^{-\frac{1}{2}\frac{(x_d-\mu_d)^2}{\sigma_d^2}} \\ &= \frac{1}{(2\Pi)^{\frac{d}{2}}\sqrt{\sigma_1^2 * ... * \sigma_d^2}} e^{-\frac{1}{2}\frac{(x_1-\mu_1)^2}{\sigma_1^2}} * ... * e^{-\frac{1}{2}\frac{(x_d-\mu_d)^2}{\sigma_d^2}} \\ &= \Big(\frac{1}{(2\Pi)}\Big)^{\frac{d}{2}} \frac{1}{\sigma_1 * ... * \sigma_d} e^{-\frac{1}{2}\frac{(x_1-\mu_1)^2}{\sigma_1^2}} * ... * e^{-\frac{1}{2}\frac{(x_d-\mu_d)^2}{\sigma_d^2}} \\ &= \Big(\frac{1}{(2\Pi)}\Big)^{\frac{1}{2}} \frac{1}{\sigma_1} * ... * \Big(\frac{1}{(2\Pi)}\Big)^{\frac{1}{2}} \frac{1}{\sigma_d} e^{-\frac{1}{2}\frac{(x_1-\mu_1)^2}{\sigma_1^2}} * ... * e^{-\frac{1}{2}\frac{(x_d-\mu_d)^2}{\sigma_d^2}} \\ &= \frac{1}{(2\Pi)^{\frac{1}{2}}\sigma_1} e^{-\frac{1}{2}\frac{(x_1-\mu_1)^2}{\sigma_1^2}} * ... * \frac{1}{(2\Pi)^{\frac{1}{2}}\sigma_d} e^{-\frac{1}{2}\frac{(x_d-\mu_d)^2}{\sigma_d^2}} \\ &= p(x_1) * ... * p(x_d) \end{aligned}$$

The fourth equality is due to the fact that the determinant of a diagonal matrix is the product of its diagonal elements. This proves the independence of the random variables.

**Solution c)**

Our training set contains $n$ points. We can, thus, write the empirical risk as the following:

$$\begin{aligned}
\hat{R} &= \frac{1}{n}\sum_{t=1}^{n} -log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sqrt{|\Sigma|}}e^{-\frac{1}{2}(x^{(t)}-\mu)^T\Sigma^{-1}(x^{(t)}-\mu)}\Big) \\
&= \frac{1}{n}\sum_{t=1}^{n} -log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sqrt{|\Sigma|}}e^{-\frac{1}{2}(x^{(t)}-\mu)^T\Sigma^{-1}(x^{(t)}-\mu)}\Big) \\
&= \frac{1}{n}\sum_{t=1}^{n}\Big(-log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sqrt{|\Sigma|}}\Big) - log\Big(e^{-\frac{1}{2}(x^{(t)}-\mu)^T\Sigma^{-1}(x^{(t)}-\mu)}\Big)\Big) \\
&= \frac{1}{n}\sum_{t=1}^{n}\Big(-log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sqrt{|\Sigma|}}\Big) + \frac{1}{2}(x^{(t)}-\mu)^T\Sigma^{-1}(x^{(t)}-\mu)\Big) \\
&= \frac{1}{n}\sum_{t=1}^{n}\Big(-log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma_1...\sigma_d}\Big) + \frac{1}{2}\Big(\frac{(x_1^{(t)}-\mu_1)^2}{\sigma_1^2} + ... + \frac{(x_d^{(t)}-\mu_d)^2}{\sigma_d^2}\Big)\Big)
\end{aligned} \tag{2}$$

See equation (1) in number 3.4.b) for a detailled explanation of the last equality.

To find the parameters, we will have to minimize the risk and get the parameters that minimize it. Thus, we will have to calculate the following:

$$\mu,\Sigma = argmin\Big(\frac{1}{n}\sum_{t=1}^{n}\Big(-log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma_1...\sigma_d}\Big) + \frac{1}{2}\Big(\frac{(x_1^{(t)}-\mu_1)^2}{\sigma_1^2} + ... + \frac{(x_d^{(t)}-\mu_d)^2}{\sigma_d^2}\Big)\Big)\Big)$$

More precisely, we will find $\mu$ by solving the following equation:

$$\frac{\partial}{\partial\mu}\Big(\frac{1}{n}\sum_{t=1}^{n}\Big(-log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma_1...\sigma_d}\Big) + \frac{1}{2}\Big(\frac{(x_1^{(t)}-\mu_1)^2}{\sigma_1^2} + ... + \frac{(x_d^{(t)}-\mu_d)^2}{\sigma_d^2}\Big)\Big)\Big) = 0$$

We will find $\Sigma$ by solving this equation:

$$\frac{\partial}{\partial\Sigma}\Big(\frac{1}{n}\sum_{t=1}^{n}\Big(-log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma_1...\sigma_d}\Big) + \frac{1}{2}\Big(\frac{(x_1^{(t)}-\mu_1)^2}{\sigma_1^2} + ... + \frac{(x_d^{(t)}-\mu_d)^2}{\sigma_d^2}\Big)\Big)\Big) = 0 \tag{3}$$

**Solution d)**

Let's first solve the equation for $\mu$:

$$\frac{\partial}{\partial\mu}\Big(\frac{1}{n}\sum_{t=1}^{n}\Big(-log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma_1...\sigma_d}\Big)+\frac{1}{2}\Big(\frac{(x_1^{(t)}-\mu_1)^2}{\sigma_1^2}+...+\frac{(x_d^{(t)}-\mu_d)^2}{\sigma_d^2}\Big)\Big)\Big)=0$$

$$\begin{bmatrix}\frac{\partial}{\partial\mu_1}\\ \vdots \\ \frac{\partial}{\partial\mu_d}\end{bmatrix}\Big(\frac{1}{n}\sum_{t=1}^{n}\Big(-log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma_1...\sigma_d}\Big)+\frac{1}{2}\Big(\frac{(x_1^{(t)}-\mu_1)^2}{\sigma_1^2}+...+\frac{(x_d^{(t)}-\mu_d)^2}{\sigma_d^2}\Big)\Big)\Big)=\begin{bmatrix}0\\ \vdots \\ 0\end{bmatrix}$$

$$\Big[\frac{1}{n}\sum_{t=1}^{n}-\frac{x_1^{(t)}-\mu_1}{\sigma_1^2},...,\frac{1}{n}\sum_{t=1}^{n}-\frac{x_d^{(t)}-\mu_d}{\sigma_d^2}\Big]^T=[0,...,0]^T$$

$$\Big[\frac{1}{\sigma_1^2}\sum_{t=1}^{n}-x_1^{(t)}+\mu_1,...,\frac{1}{\sigma_d^2}\sum_{t=1}^{n}-x_d^{(t)}+\mu_d\Big]^T=[0,...,0]^T$$

$$\Big[\sum_{t=1}^{n}-x_1^{(t)}+\mu_1,...,\sum_{t=1}^{n}-x_d^{(t)}+\mu_d\Big]^T=[0,...,0]^T$$

$$\Big[n\mu_1-\sum_{t=1}^{n}x_1^{(t)},...,n\mu_d-\sum_{t=1}^{n}x_d^{(t)}\Big]^T=[0,...,0]^T$$

$$\iff n\mu_i=\sum_{t=1}^{n}x_i^{(t)}\text{ for all }i=1,...d$$

$$\iff \mu_i=\frac{1}{n}\sum_{t=1}^{n}x_i^{(t)}\text{ for all }i=1,...d$$

Let's now solve the equation for $\Sigma$:

$$\frac{\partial}{\partial\Sigma}\Big(\frac{1}{n}\sum_{t=1}^{n}\Big(-log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma_1...\sigma_d}\Big)+\frac{1}{2}\Big(\frac{(x_1^{(t)}-\mu_1)^2}{\sigma_1^2}+...+\frac{(x_d^{(t)}-\mu_d)^2}{\sigma_d^2}\Big)\Big)\Big)=0$$

Because our variance-covariance matrix is diagonal, calculating $\frac{\partial}{\partial\Sigma}$ is the same as computing $\begin{bmatrix}\frac{\partial}{\partial\sigma_1}\\ \vdots \\ \frac{\partial}{\partial\sigma_d}\end{bmatrix}$. Thus we need to calculate:

$$\begin{bmatrix} \frac{\partial}{\partial \sigma_1} \\ \vdots \\ \frac{\partial}{\partial \sigma_d} \end{bmatrix} \Big(\frac{1}{n}\sum_{t=1}^{n}\Big( - log\Big(\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma_1 ... \sigma_d}\Big) + \frac{1}{2}\Big(\frac{(x_1^{(t)} - \mu_1)^2}{\sigma_1^2} + ... + \frac{(x_d^{(t)} - \mu_d)^2}{\sigma_d^2}\Big)\Big)\Big) = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\Big[\frac{1}{n}\sum_{t=1}^{n}\Big( - \frac{1}{\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma_1 ... \sigma_d}}\Big( - \frac{1}{(2\Pi)^{\frac{d}{2}}\sigma_1^2 ... \sigma_d}\Big) - \frac{(x_1^{(t)} - \mu_1)^2}{\sigma_1^3}\Big), ...,$$

$$\frac{1}{n}\sum_{t=1}^{n}\Big( - \frac{1}{\frac{1}{(2\Pi)^{\frac{d}{2}}\sigma_1 ... \sigma_d}}\Big( - \frac{1}{(2\Pi)^{\frac{d}{2}}\sigma_1 ... \sigma_d^2}\Big) - \frac{(x_d^{(t)} - \mu_d)^2}{\sigma_d^3}\Big)\Big]^T = [0, ..., 0]^T$$

$$\Big[\sum_{t=1}^{n}\Big(\frac{(2\Pi)^{\frac{d}{2}}\sigma_1 ... \sigma_d}{(2\Pi)^{\frac{d}{2}}\sigma_1^2 ... \sigma_d} - \frac{(x_1^{(t)} - \mu_1)^2}{\sigma_1^3}\Big), ..., \sum_{t=1}^{n}\Big(\frac{(2\Pi)^{\frac{d}{2}}\sigma_1 ... \sigma_d}{(2\Pi)^{\frac{d}{2}}\sigma_1 ... \sigma_d^2} - \frac{(x_d^{(t)} - \mu_d)^2}{\sigma_d^3}\Big)\Big]^T = [0, ..., 0]^T$$

$$\Big[\sum_{t=1}^{n}\Big(\frac{1}{\sigma_1} - \frac{(x_1^{(t)} - \mu_1)^2}{\sigma_1^3}\Big), ..., \sum_{t=1}^{n}\Big(\frac{1}{\sigma_d} - \frac{(x_d^{(t)} - \mu_d)^2}{\sigma_d^3}\Big)\Big]^T = [0, ..., 0]^T$$

$$\Big[\frac{n}{\sigma_1} - \sum_{t=1}^{n}\frac{(x_1^{(t)} - \mu_1)^2}{\sigma_1^3}, ..., \frac{n}{\sigma_d} - \sum_{t=1}^{n}\frac{(x_d^{(t)} - \mu_d)^2}{\sigma_d^3}\Big]^T = [0, ..., 0]^T$$

$$\iff \frac{n}{\sigma_i} = \sum_{t=1}^{n}\frac{(x_i^{(t)} - \mu_i)^2}{\sigma_i^3} \text{ for all } i = 1, ...d$$

$$\iff \frac{n}{\sigma_i} = \frac{1}{\sigma_i^3}\sum_{t=1}^{n}(x_i^{(t)} - \mu_i)^2 \text{ for all } i = 1, ...d$$

$$\iff n\sigma_i^2 = \sum_{t=1}^{n}(x_i^{(t)} - \mu_i)^2 \text{ for all } i = 1, ...d$$

$$\iff \sigma_i^2 = \frac{1}{n}\sum_{t=1}^{n}(x_i^{(t)} - \mu_i)^2 \text{ for all } i = 1, ...d$$

# Problem 4

September 28, 2018

## 4 Practical part: density estimation

```
In [1]: import numpy as np #for math formulas
        #to enable the graph to appear alongside the output in the notebook
        %pylab inline
        import pylab       #for the graphs
        import random      #for constructing random sets
```

```
Populating the interactive namespace from numpy and matplotlib
```

### 4.1 Problem

Implement a diagonal Gaussian parametric density estimator. It will have to work for data of arbitrary dimension $d$. As seen in the labs, it should have a train() method to learn the parameters and a method predict() which calculates the log density.

#### Solution

The method trainDGaussian() takes as an argument the set on which we wish to train the parameters $\mu$ (the mean) and $\sigma^2$ (the variance). It calculates the best parameters based on the formulas that we derived in 3.4.d). It returns $\mu$ (the mean) and $\sigma$ (the standard deviation instead of the variance for convenient purposes).

```
In [2]: def trainDGaussian(train_set):
            numPoints = train_set.shape[0]                                  #gives
            mu = (1/numPoints)*np.sum(train_set,axis=0)                     #calcul
            sigma = ((1/numPoints)*np.sum((train_set-mu)**2,axis=0))**(1/2) #calcul
            return mu,sigma                                                 #return
```

The method predictDGaussian() takes as arguments the point $x$ for which we want to find its log density, the estimated mean $\mu$ and the estimated standard variation $\sigma$. It calculates the log density of point $x$ based on the formula that we wrote in 3.4.a). It returns the log density of $x$.

```
In [3]: def predictDGaussian(x,mu,sigma):
            numDim = len(x)                                         #gives the di
            piPart = 1/(((2*np.pi)**(numDim/2))*np.prod(sigma))     #calculates t
            exponential = np.e**np.sum(((-1/2)*(x-mu)**2)/(sigma**2)) #calculates t
            return np.log(piPart*exponential)                       #returns the
```

### 4.2 Problem

Implement a Parzen density estimator with an isotropic Gaussian kernel. It will have to work for data of arbitrary dimension $d$. Likewise it should have a train() method and a predict() method that computes the log density.

#### Solution

The method trainParzen() takes as an argument the set on which we wish to train our model. It calculates nothing because there are no parameters to be trained. The Parzen Gaussian approach has only a hyperparameter that will be found in another function later in this homework. As said in question 3.2.a) the training phase consists of storing the data. As we did not make a classifier it returns the train set as it is.

```
In [4]: def trainParzen(train_set):
            return train_set
```

The method predictParzen() takes as arguments the point $x$ for which we want to find its log density, the training set and the hyperparameter $\sigma$. It calculates the log density of point $x$ based on the formula that we wrote in 3.2.b). It returns the log density of $x$.

```
In [5]: def predictParzen(x,train_set,sigma):
            numDim = len(x)                                           #gives the dimensi
            numPoints = train_set.shape[0]                            #gives the number 
            sumation = 0                                              #initializes the s
            piPart = 1/(((2*np.pi)**(numDim/2))*(sigma**numDim))  #calculates the Pi
            for i in range(numPoints):
                exponential = np.e**(((-1/2)*np.sum((x-train_set[i])**2))/(sigma**2
                sumation += piPart*exponential                        #calculates the su
            return np.log((1/numPoints)*sumation)                     #returns the log d
```

## 4.3 Problem

1D densities: From the Iris dataset examples, choose a subset corresponding to one of the classes (of your choice), and one of the characteristic features, so that we will be in dimension $d = 1$ and produce a single graph (using the plot function) including:

(a) the data points of the subset (displayed on the x axis).

(b) a plot of the density estimated by your parametric Gaussian estimator.

(c) a plot of the density estimated by the Parzen estimator with a hyper-parameter $\sigma$ (standard deviation) too small.

(d) a plot of the density estimated by the Parzen estimator with the hyper-parameter $\sigma$ being a little too big.

(e) a plot of the density estimated by the Parzen estimator with the hyper-parameter $\sigma$ that you consider more appropriate. Use a different color for each plot, and provide your graph with a clear legend.

(f) Explain how you chose your hyper-parameter.

**Solutions for a, b, c, d, e, and f)**

**Remark:**

- The solutions for question 4.3 are not in order because our results were better explained in another way.

The class that we decided to work on is class 1 (the first 50 entries in the Iris dataset example) and we chose to study the first feature of this class.

```
In [6]: iris = np.loadtxt("iris.txt") #loads the data
        dataChosen = iris[:50,0]      #first class and first feature chosen
        print(dataChosen)
```

```
[ 5.1  4.9  4.7  4.6  5.   5.4  4.6  5.   4.4  4.9  5.4  4.8  4.8  4.3  5.8
  5.7  5.4  5.1  5.7  5.1  5.4  5.1  4.6  5.1  4.8  5.   5.   5.2  5.2  4.7
  4.8  5.4  5.2  5.5  4.9  5.   5.5  4.9  4.4  5.1  5.   4.5  4.4  5.   5.1
  4.8  5.1  4.6  5.3  5. ]
```

Before producing the graph, let's find the parameters $\mu_{DGaussian}$ and $\sigma_{DGaussian}$ for the parametric diagonal Gaussian approach and decide the value that the hyperparameter of the Parzen Gaussian approach should take (the good $\sigma_{Parzen}$).

**Finding the parameters for the parametric diagonal Gaussian approach**

```
In [7]: muDGaussian, sigmaDGaussian = trainDGaussian(dataChosen)                    #We
        print('The trained value of the parameter mu is:', muDGaussian)
        print('The trained value of the parameter sigma is:', sigmaDGaussian)
```

```
The trained value of the parameter mu is: 5.006
The trained value of the parameter sigma is: 0.348946987378
```

**Remark:**

- In this homework, it is not necessary to create a test set as we are not asked to analyse the performances of the approaches that we implemented. We are only asked to plot the functions on graphs. As we do not have a lot of data, it is convenient to put this set aside for now.

**Finding the hyperparameter for the Parzen Gaussian approach**

The method findSigma() takes as arguments the data points (inputPoints) on which we want to find the prediction errors and the training set. The function calculates the empirical risk of the Parzen approach for a number of fixed $\sigma$ ranging from 0.02 to 1. It returns the value of the hyperparameter that minimizes the risk.

The loss function that was taken to minimize the empirical risk is the negative log of the Parzen Gaussian density.

```
In [8]: def findSigma(inputPoints,train_set):
            numPoints = inputPoints.shape[0]                     #gives the number of po
            empiricalRiskTable = []                              #initializes a table th
            for sigmaParzenTest in np.arange(0.02,1,0.02):       #the values of the hype
                empiricalRisk = 0                                #initializes the empiri
                for i in range(numPoints):
                    if(isinstance(inputPoints[i], float)):
                        empiricalRisk += -predictParzen([inputPoints[i]],train_set,
                    else:
                        empiricalRisk += -predictParzen(inputPoints[i],train_set,si
                empiricalRiskTable.append(empiricalRisk/numPoints)      #puts the r
            sigmaMin = (np.argmin(np.array(empiricalRiskTable))+1)*0.02 #finds the
            return sigmaMin                                             #returns th
```

To find the best sigma, we will have to separate our data set into two sets: the training set and the validation set (no test set needed as explained above). These two sets are necessary for finding a good $\sigma$. Indeed, we need to make sure that our function fits well data points that are not in the training set. We decided that our training set would have 35 values out of the 50 that we can work with and our validation set would have 15.

Depending on content of these two sets, the sigma that minimizes the empirical risk might be different. The chance of that happening is relatively high since we have a small number of data points to work with. We, thus, decided to find the best sigma on 100 random composition of the training and validation sets. Our final best sigma is the mean of these 100 values.

```
In [9]: def bestAverageSigma(data_set):
            numRepeat = 100                         #gives the number of times that we wa
            meanValid = np.zeros(numRepeat)         #initializes the vector meanValid tha
            for j in range(numRepeat):
                inds = list(range(data_set.shape[0]))    #gives a list of the indice
                random.shuffle(inds)                     #shuffles the list of indic
                train_inds = inds[:35]                   #takes the first 35 shuffle
                valid_inds = inds[35:]                   #takes the last 15 shuffled
                train_set = data_set[train_inds]         #makes the training set
                valid_set = data_set[valid_inds]         #makes the validation set
                meanValid[j] = findSigma(valid_set,train_set) #puts in the table th
            print("the best sigma for the validation set is", np.mean(meanValid))
            return np.mean(meanValid)                         #returns the best sig

In [10]: goodSigma = bestAverageSigma(dataChosen)

the best sigma for the validation set is 0.1478
```

Now let's plot the estimated densities onto a graph.
**Remark:**

- Because our chosen data set contains several points of same values, we will put a shading effect on the points. That way, we will have an idea of the density of our points. The more points there will be on the same spot, the darker they will appear. It's the alpha argument in the plot function that allows this to happen.

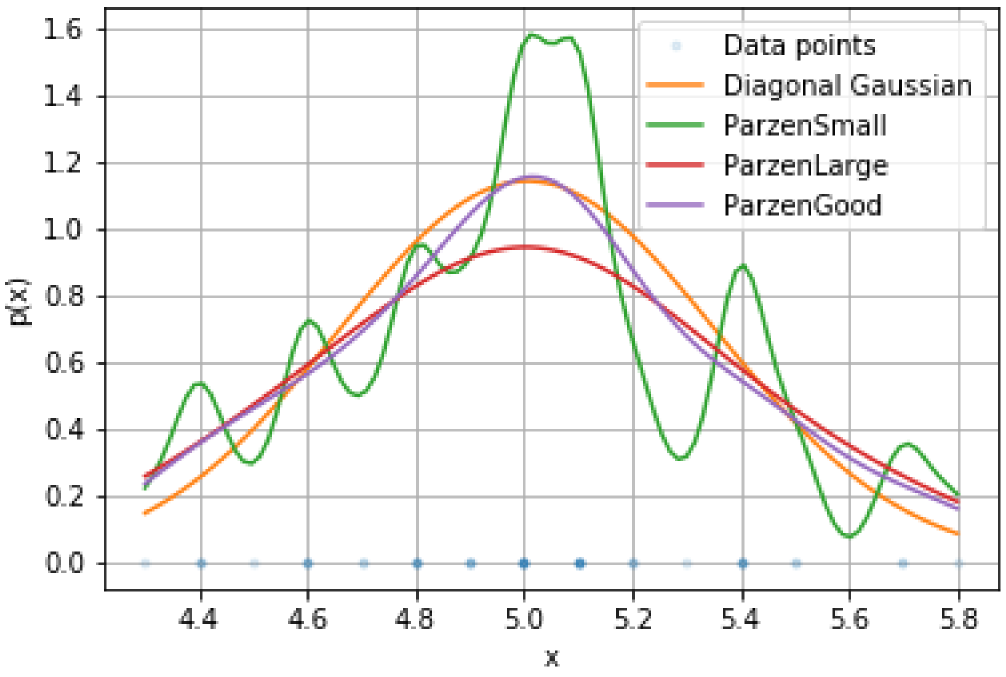
```
In [11]: #Plotting the chosen data points
         supportYaxis = np.zeros(50) #that will display the points on the x axis
         pylab.plot(dataChosen,supportYaxis,'.',alpha=0.125) #We plot the points

         #Plotting the predicted density function of the parametric diagonal Gaussi
         supportXaxis = linspace(4.3,5.8,100)
         density = [np.e**predictDGaussian([supportXaxis[i]],muDGaussian,sigmaDGaus
         pylab.plot(supportXaxis,density)

         #Plotting the predicted density function of the Parzen Gaussian approach
         sigmaParzenChosen = [goodSigma-0.1,goodSigma+0.1,goodSigma]
         for sigmaParzen in sigmaParzenChosen:
             densityParzen = [np.e**predictParzen([supportXaxis[i]],dataChosen,sigm
             pylab.plot(supportXaxis,densityParzen)

         #Graph specificities
         pylab.grid(True)
         pylab.xlabel('x')
         pylab.ylabel('p(x)')
         pylab.legend(('Data points', 'Diagonal Gaussian','ParzenSmall','ParzenLarg

Out[11]: <matplotlib.legend.Legend at 0x4a51663cf8>
```



**Remarks:**

1. The parametric Gaussian curve is well plotted because the mean corresponds to the highest point on the curve.

2. The curve drawn by the Parzen Gaussian approach with a small sigma is definitely too small. We see that the curve reacts too much to our training data (overfit).
3. The curve drawn by the Parzen Gaussian approach with a large sigma is too big. We see that the curve does not go high enough (underfit).
4. The curve drawn by the Parzen Gaussian approach with a good sigma resembles the parametric Gaussian one. These two curves seem to really catch the shape of the density.

### 4.4 Problem

2D densities: Now add a second characteristic feature of Iris, in order to have entries in $d = 2$ and produce 4 plots, each displaying the points of the subset of the data (with the plot function ), and the contour lines of the density estimated (using the contour function):

(a) by the diagonal Gaussian parametric estimator.

(b) by the Parzen estimator with the hyper-parameter $\sigma$ (standard deviation ) being too small.

(c) by the Parzen estimator with the hyper-parameter $\sigma$ being a little too big.

(d) by the Parzen estimator with the hyper-parameter $\sigma$ that you consider more appropriate.

(e) Explain how you chose your hyper-parameter $\sigma$

**Solutions for a,b,c,d and e)**

**Remark:**

- The solutions for question 4.4 are not in order because our results were better explained in another way.

To do this problem, we chose to add the second feature of our iris dataset.

```
In [12]: dataChosen2D = iris[:50,:2] #first class and first and second feature chos
         print(dataChosen2D)
```

```
[[ 5.1  3.5]
 [ 4.9  3. ]
 [ 4.7  3.2]
 [ 4.6  3.1]
 [ 5.   3.6]
 [ 5.4  3.9]
 [ 4.6  3.4]
 [ 5.   3.4]
 [ 4.4  2.9]
 [ 4.9  3.1]
 [ 5.4  3.7]
 [ 4.8  3.4]
 [ 4.8  3. ]
 [ 4.3  3. ]
 [ 5.8  4. ]
 [ 5.7   4.4]
 [ 5.4   3.9]
 [ 5.1   3.5]
 [ 5.7   3.8]
 [ 5.1   3.8]
 [ 5.4   3.4]
 [ 5.1   3.7]
 [ 4.6   3.6]
 [ 5.1   3.3]
 [ 4.8   3.4]
 [ 5.    3. ]
 [ 5.    3.4]
 [ 5.2   3.5]
 [ 5.2   3.4]
 [ 4.7   3.2]
 [ 4.8   3.1]
 [ 5.4   3.4]
 [ 5.2   4.1]
 [ 5.5   4.2]
 [ 4.9   3.1]
 [ 5.    3.2]
 [ 5.5   3.5]
 [ 4.9   3.6]
 [ 4.4   3. ]
 [ 5.1   3.4]
 [ 5.    3.5]
 [ 4.5   2.3]
 [ 4.4   3.2]
 [ 5.    3.5]
 [ 5.1   3.8]
 [ 4.8   3. ]
 [ 5.1   3.8]
 [ 4.6   3.2]
 [ 5.3   3.7]
 [ 5.    3.3]]
```

As we did for the exercise in 1D, let's find the parameters $\mu_{DGaussian}$ and $\sigma_{DGaussian}$ for the parametric diagonal Gaussian approach and decide the value that the hyperparameter of the Parzen Gaussian approach should take (the good $\sigma_{parzen}$).

**Finding the parameters for the 2D parametric diagonal Gaussian approach**

```
In [13]: muDGaussian2D, sigmaDGaussian2D = trainDGaussian(dataChosen2D)          #
         print('The trained value of the parameter mu is:', muDGaussian2D)
         print('The trained value of the parameter sigma is:', sigmaDGaussian2D)
```

```
The trained value of the parameter mu is: [ 5.006  3.428]
The trained value of the parameter sigma is: [ 0.34894699  0.37525458]
```

**Remark:**

- As explained in problem 4.3) no test set is needed in this homework.

**Finding the hyperparameter for the 2D Parzen Gaussian approach**

As the methods bestAverageSigma() and findSigma() works for any dimension, we can call it for our problem in 2D.

```
In [14]: goodSigma2D = bestAverageSigma(dataChosen2D)
```

```
the best sigma for the validation set is 0.1956
```

The method combine is a method that was directly taken from the lab session 4. It returns a list of all combinations of argument sequences. It is necessary for doing the 2D plot. For example: combine((1,2),(3,4)) returns [[1, 3], [1, 4], [2, 3], [2, 4]]

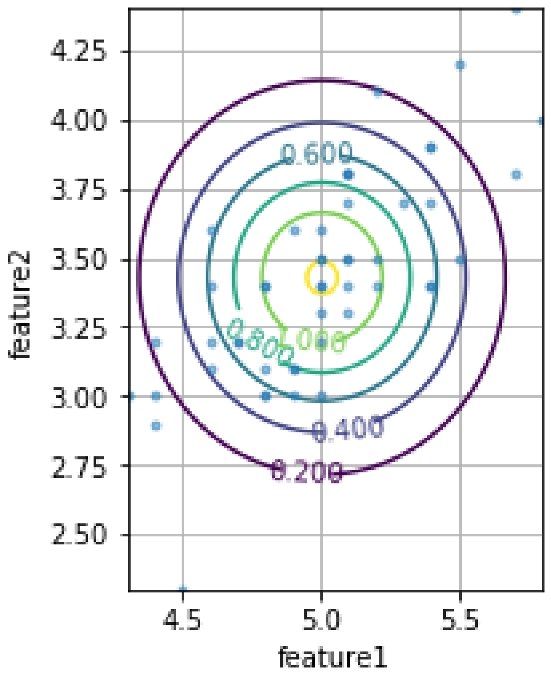
```
In [15]: def combine(*seqin):
             def rloop(seqin,listout,comb):
                 '''recursive looping function'''
                 if seqin:                       # any more sequences to process?
                     for item in seqin[0]:
                         newcomb=comb+[item]     # add next item to current comb
                         # call rloop w/ rem seqs, newcomb
                         rloop(seqin[1:],listout,newcomb)
                 else:                           # processing last sequence
                     listout.append(comb)        # comb finished, add to list
             listout=[]                      # listout initialization
             rloop(seqin,listout,[])         # start recursive process
             return listout
```

We display the points and the contour lines of the density estimated by the diagonal parametric Gaussian. As in question 4.3) we will add a shading effect to the points:

```
In [16]: #Plotting the chosen 2D data points
         pylab.plot(dataChosen2D[:,0],dataChosen2D[:,1],'.',alpha=0.5)

         #Plotting the level curves of the predicted density function
         feature1 = np.linspace(4.3, 5.8, 100)
         feature2 = np.linspace(2.3, 4.4, 100)
         supportXY2D = np.array(combine(feature1,feature2))
         density2D = np.array([np.e**predictDGaussian(supportXY2D[i],muDGaussian2D,
         density2DReshape = np.transpose(density2D.reshape(100,100))
         contour = pylab.contour(feature1, feature2, density2DReshape)
         pylab.clabel(contour, inline=1, fontsize=10)
```

```
        #Graph specificities
        pylab.grid(True)                              #adds a grid to the graph
        pylab.xlabel('feature1')                      #adds the label for the x axi
        pylab.ylabel('feature2')                      #adds the label for the y axi
        pylab.gca().set_aspect('equal')               #makes the axis of the same s
```



**Remarks**

1. The graph is well plotted. Indeed, the mean that we found is in the center of the graph and we can see that the standard deviation is bigger for feature 2 than feature 1 by the elongation of the level curves in the direction of the y axis.
2. The level curves are not affected by noise. But they don't seem to catch the shape of the distribution. The points are distributed diagonally, but the level curves seem to think that they are not. This is caused by the choice of using a parametric diagonal Gaussian approach for which the correlation between the features is equal to 0. If it was not, the level curves could have been more elliptic and better match the distribution.

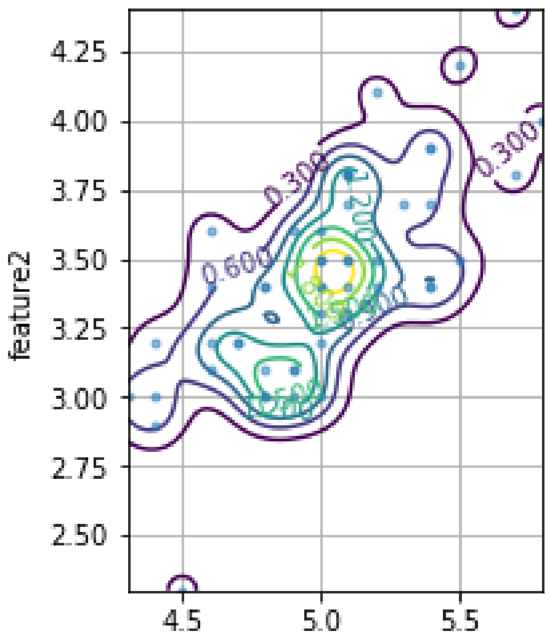
We now display the points and the contour lines of the density estimated by the Parzen Gaussian with small $\sigma$ (see code block for the plot of the parametric diagonal Gaussian for more detailed comments in the following code):

```
In [17]: #Plotting the chosen 2D data points
         pylab.plot(dataChosen2D[:,0],dataChosen2D[:,1],'.',alpha=0.5)

         #Plotting the level curves of the predicted density function
```

```
        sigmaParzen2DSmall = goodSigma2D-0.1
        density2DParzenSmall = np.array([np.e**predictParzen(supportXY2D[i],dataCh
        density2DReshapeParzenSmall = np.transpose(density2DParzenSmall.reshape(10
        contour = pylab.contour(feature1, feature2, density2DReshapeParzenSmall)
        pylab.clabel(contour, inline=1, fontsize=10)

        #Graph specificities
        pylab.grid(True)
        pylab.ylabel('feature1')
        pylab.ylabel('feature2')
        pylab.gca().set_aspect('equal', adjustable='box')
```



**Remark:**

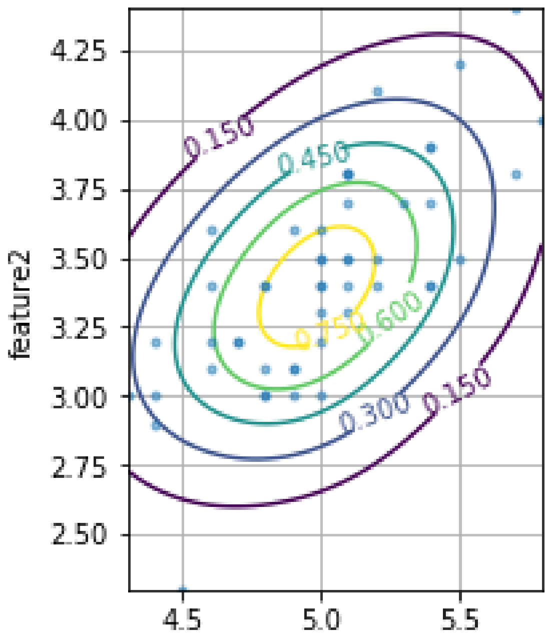
- The level curves drawn by the Parzen Gaussian approach with a small $\sigma$ shows that it is definitely too small. We see that the curve reacts too much to our training data (overfit).

We now display the points and the contour lines of the density estimated by the Parzen Gaussian with large $\sigma$ (see code block for the plot of the parametric diagonal Gaussian for more detailed comments in the following code):

```
In [18]: #Plotting the chosen 2D data points
        pylab.plot(dataChosen2D[:,0],dataChosen2D[:,1],'.',alpha=0.5)

        #Plotting the level curves of the predicted density function
        sigmaParzen2DLarge = goodSigma2D+0.1
        density2DParzenLarge = np.array([np.e**predictParzen(supportXY2D[i],dataCh
        density2DReshapeParzenLarge = np.transpose(density2DParzenLarge.reshape(10
        contour = pylab.contour(feature1, feature2, density2DReshapeParzenLarge)
        pylab.clabel(contour, inline=1, fontsize=10)

        #Graph specificities
        pylab.grid(True)
        pylab.ylabel('feature1')
        pylab.ylabel('feature2')
        pylab.gca().set_aspect('equal', adjustable='box')
```



**Remark:**

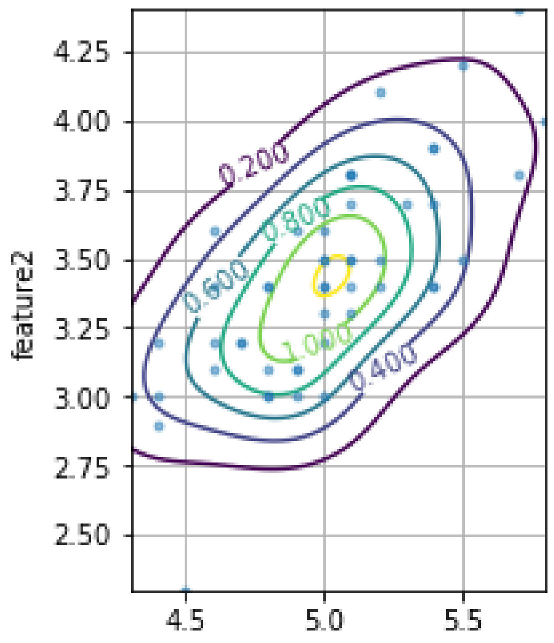
- The level curves drawn by the Parzen Gaussian approach with a large $\sigma$ shows that it is too large. We see that the centered curve has a relatively small value which means that, in 3D, the density function does not go very high (not high enough) (underfit).

We now display the points and the contour lines of the density estimated by the Parzen Gaussian with good $\sigma$ (see code block for the plot of the parametric diagonal Gaussian for more detailed comments in the following code):

```
In [19]: #Plotting the chosen 2D data points
         pylab.plot(dataChosen2D[:,0],dataChosen2D[:,1],'.',alpha=0.5)

         #Plotting the level curves of the predicted density function
         sigmaParzen2DGood = goodSigma2D
         density2DParzenGood = np.array([np.e**predictParzen(supportXY2D[i],dataCho
         density2DReshapeParzenGood = np.transpose(density2DParzenGood.reshape(100,
         contour = pylab.contour(feature1, feature2, density2DReshapeParzenGood)
         pylab.clabel(contour, inline=1, fontsize=10)

         #Graph specificities
         pylab.grid(True)
         pylab.ylabel('feature1')
         pylab.ylabel('feature2')
         pylab.gca().set_aspect('equal', adjustable='box')
```



**Remark:**

- The level curves are not affected by noise. They seem to really catch the shape of the distribution and the density of the points.

```
In [ ]:
```