FYS-2021: Machine Learning.
Assignment 2

Artemii Gribkov

Contents

1 Introduction **1**

2 Problem 1 **1**
2.1 Problem 1a . . . 2
2.2 Problem 1b . . . 6
2.3 Problem 1c . . . 6
2.4 Problem 1d . . . 6
2.5 Problem 1e . . . 7
2.6 Problem 1f . . . 7
2.7 Problem 1g . . . 8

3 Problem 2 **9**
3.1 Problem 2a . . . 9
3.2 Problem 2b . . . 10
3.3 Problem 2c . . . 12
3.4 Problem 2d . . . 13

4 Conclusion **13**

5 References **13**

1 Introduction

The second assignment in Machine Learning consists of two parts. In part one our main focus is a theoretical understanding of the course, namely linear regression loss functions, and some practical focus, namely practice with the learning process by gradient descent on the loss surface. In the second part, our focus is on working and presenting the results. For example, we will use our previous experience with training and test sets to show the result with further explanation or discussion.

2 Problem 1

To answer the first part of assignment 2, we need to introduce the linear regression line and chain rule, since both will be used to explain or solve the sub-problems.

The linear regression equation is expressed as:

$$\hat{y} = w_1 x + w_0 \tag{1}$$

where:

- x is scalar since this problem set has only one feature.
- $\hat{y}$ is predicted value.

- w_1 is the slope of the line.
- w_0 is the y-intercept.

The chain rule for the derivation of composite function is expressed as:

$$h'(x) = (f \circ g)'(x) = f'(g(x)) \cdot g'(x) \tag{2}$$

where $h(x)$ is composite function.

2.1 Problem 1a

Several types of loss functions are used for linear regression but the most common are:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Huber Loss / Smooth Mean Absolute Error

Mean Squared Error (MSE)

Mean Squared Error (MSE) calculates how far off actual values are from predicted values by averaging the squared differences. The following mathematical formula represents the MSE function:

$$L_{\text{MSE}}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3}$$

where:

- n is the number of data points,
- y_i is the actual value for the i-th data point,
- $\hat{y}_i$ is the predicted value for the i-th data point.

Derivation of the gradient of MSE loss function for $\frac{\partial L_{\text{MSE}}}{\partial w_1}$ and $\frac{\partial L_{\text{MSE}}}{\partial w_0}$ (each weight parameter) can be achieved as follows:

For w_0:

$$\frac{\partial L_{\text{MSE}}}{\partial w_0} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_0} (y_i - \hat{y}_i)^2 \tag{4}$$

$$= \frac{1}{n} \sum_{i=1}^{n} -2(y_i - \hat{y}_i) \frac{\partial \hat{y}_i}{\partial w_0} \tag{5}$$

For $\hat{y}_i = w_0 + w_1 x_i$, we have:

$$\frac{\partial \hat{y}_i}{\partial w_0} = 1 \tag{6}$$

Thus, substituting this into our expression gives:

$$\frac{\partial L_{\text{MSE}}}{\partial w_0} = \frac{1}{n} \sum_{i=1}^{n} -2(y_i - \hat{y}_i) \cdot 1 \tag{7}$$

$$= -\frac{2}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i) \tag{8}$$

For w_1:

$$\frac{\partial L_{\text{MSE}}}{\partial w_1} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial w_1}(y_i - \hat{y}_i)^2 \tag{9}$$

$$= \frac{1}{n}\sum_{i=1}^{n} -2(y_i - \hat{y}_i)\frac{\partial \hat{y}_i}{\partial w_1} \tag{10}$$

For $\hat{y}_i = w_0 + w_1 x_i$, we find:

$$\frac{\partial \hat{y}_i}{\partial w_1} = x_i \tag{11}$$

Thus, substituting this into our expression gives:

$$\frac{\partial L_{\text{MSE}}}{\partial w_1} = \frac{1}{n}\sum_{i=1}^{n} -2(y_i - \hat{y}_i)x_i \tag{12}$$

$$= -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)x_i \tag{13}$$

The main advantage of the MSE is differentiability which means it is differentiable everywhere) and this function works well with Large Errors because of squaring. One of the main disadvantages is that MSE is sensitive to the outliers that heavily influence the function.

Mean Absolute Error(MAE)

Mean Absolute Error calculates the average of the absolute differences between actual values and predicted values. The following mathematical formula represents the MAE function:

$$L_{\text{MAE}}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{14}$$

Derivation of the gradient of MAE loss function for $\frac{\partial L_{\text{MAE}}}{\partial w_1}$ and $\frac{\partial L_{\text{MAE}}}{\partial w_0}$ (each weight parameter) can be achieved as follows:

$$|y_i - \hat{y}_i| = \begin{cases} y_i - \hat{y}_i & \text{if } y_i - \hat{y}_i \geq 0 \\ -(y_i - \hat{y}_i) & \text{if } y_i - \hat{y}_i < 0 \end{cases} \tag{15}$$

For w_0:

$$\frac{\partial L_{\text{MAE}}}{\partial w_0} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial w_0}|y_i - \hat{y}_i| \tag{16}$$

Using the chain rule:

$$\frac{\partial}{\partial w_0}|y_i - \hat{y}_i| = \begin{cases} -1 & \text{if } y_i - \hat{y}_i > 0 \\ 1 & \text{if } y_i - \hat{y}_i < 0 \end{cases} \tag{17}$$

Thus we have:

$$\frac{\partial L_{\text{MAE}}}{\partial w_0} = \frac{1}{n}\sum_{i=1}^{n}\begin{cases} -1 & \text{if } y_i - \hat{y}_i > 0 \\ 1 & \text{if } y_i - \hat{y}_i < 0 \end{cases} \tag{18}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\text{sgn}(y_i - \hat{y}_i) \tag{19}$$

where $\text{sgn}(x)$ is the sign function that returns -1, 0, or 1.
For w_1:

$$\frac{\partial L_{\text{MAE}}}{\partial w_1} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial w_1}|y_i - \hat{y}_i| \tag{20}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial w_1}|y_i - (w_0 + w_1 x_i)| \tag{21}$$

Using the chain rule:

$$\frac{\partial}{\partial w_1}|y_i - \hat{y}_i| = -x_i \cdot \text{sgn}(y_i - \hat{y}_i) \tag{22}$$

Thus, we have:

$$\frac{\partial L_{\text{MAE}}}{\partial w_1} = \frac{1}{n}\sum_{i=1}^{n} -x_i \cdot \text{sgn}(y_i - \hat{y}_i) \tag{23}$$

$$= -\frac{1}{n}\sum_{i=1}^{n} x_i \cdot \text{sgn}(y_i - \hat{y}_i) \tag{24}$$

MAE at the same time is non-differentiable and works not well with large errors but works better with outliers than MSE and one more advantage is that this function treats all errors equally.

Huber Loss

Huber Loss is relevant because this function is a combination of the advantages of MSE and MAE functions. For small errors, it behaves like MSE and for large errors, it behaves like MAE. The following mathematical formula represents the Huber Loss function:

$$L_{Huber}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2, & \text{for } |y_i - \hat{y}_i| \leq \delta \\ \delta|y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & otherwise. \end{cases} \tag{25}$$

where:

- δ is a hyper-parameter that defines how small an error has to be or in other words, a hyper-parameter that controls the behavior (MSE or MAE case).

Derivation of the gradient of Huber Loss function for $\frac{\partial L_{\text{Huber}}}{\partial w_1}$ and $\frac{\partial L_{\text{Huber}}}{\partial w_0}$ (each weight parameter) can be achieved as follows:

For w_0:
Case 1: $|y_i - \hat{y}_i| \leq \delta$

$$L_{\text{Huber}}(y, \hat{y}) = \frac{1}{2}(y_i - \hat{y}_i)^2 \tag{26}$$

Calculating the derivative for w_0:

$$\frac{\partial L_{\text{Huber}}}{\partial w_0} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial w_0}\left(\frac{1}{2}(y_i - \hat{y}_i)^2\right) \tag{27}$$

$$= \frac{1}{n}\sum_{i=1}^{n} -(y_i - \hat{y}_i)\frac{\partial \hat{y}_i}{\partial w_0} \tag{28}$$

$$= \frac{1}{n}\sum_{i=1}^{n} -(y_i - \hat{y}_i) \cdot 1 \tag{29}$$

$$= -\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i) \tag{30}$$

Case 2: $|y_i - \hat{y}_i| > \delta$

$$L_{Huber}(y, \hat{y}) = \delta|y_i - \hat{y}_i| - \frac{1}{2}\delta^2 \tag{31}$$

Calculating the derivative:

$$\frac{\partial L_{\text{Huber}}}{\partial w_0} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial w_0}\left(\delta(|y_i - \hat{y}_i| - \frac{1}{2}\delta)\right) \tag{32}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \delta \cdot \text{sgn}(y_i - \hat{y}_i)\frac{\partial \hat{y}_i}{\partial w_0} \tag{33}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \delta \cdot \text{sgn}(y_i - \hat{y}_i) \cdot 1 \tag{34}$$

$$= \frac{\delta}{n}\sum_{i=1}^{n} \text{sgn}(y_i - \hat{y}_i) \tag{35}$$

Combining both cases, we have:

$$\frac{\partial L_{\text{Huber}}}{\partial w_0} = \begin{cases} -\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \frac{\delta}{n}\sum_{i=1}^{n}\text{sgn}(y_i - \hat{y}_i) & \text{if } |y_i - \hat{y}_i| > \delta \end{cases} \tag{36}$$

For w_1:
Case 1: $|y_i - \hat{y}_i| \leq \delta$

$$\frac{\partial L_{\text{Huber}}}{\partial w_1} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial w_1}\left(\frac{1}{2}(y_i - \hat{y}_i)^2\right) \tag{37}$$

$$= \frac{1}{n}\sum_{i=1}^{n} -(y_i - \hat{y}_i)\frac{\partial \hat{y}_i}{\partial w_1} \tag{38}$$

$$= \frac{1}{n}\sum_{i=1}^{n} -(y_i - \hat{y}_i)x_i \tag{39}$$

$$= -\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)x_i \tag{40}$$

Case 2: $|y_i - \hat{y}_i| > \delta$ The derivative becomes:

$$\frac{\partial L_{\text{Huber}}}{\partial w_1} = \frac{1}{n}\sum_{i=1}^{n} \delta \cdot \text{sgn}(y_i - \hat{y}_i)\frac{\partial \hat{y}_i}{\partial w_1} \tag{41}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \delta \cdot \text{sgn}(y_i - \hat{y}_i)x_i \tag{42}$$

Combining both cases gives:

$$\frac{\partial L_{\text{Huber}}}{\partial w_1} = \begin{cases} -\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)x_i & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \frac{\delta}{n}\sum_{i=1}^{n}\text{sgn}(y_i - \hat{y}_i)x_i & \text{if } |y_i - \hat{y}_i| > \delta \end{cases} \tag{43}$$

Huber Loss takes all advantages of MSE and MAE but its main disadvantage is the choice of suitable δ.

2.2 Problem 1b

There are several reasons why smooth loss functions are preferred when using gradient descent:

- **Differentiability**. The Smooth loss functions have better differentiability, which leads to continuous gradient computation, which leads to good optimization.
- **Stable and Smooth Updates**. The Smooth loss functions provide stable updates of parameters.
- **No abrupt changes** For Smooth loss functions the gradients vary continuously without any abrupt changes

As an example, we can compare MSE and MAE Loss functions, that were described in subtask 1a.

$$L_{\text{MSE}}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{44}$$

$$L_{\text{MAE}}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{45}$$

One of the biggest problems of MAE is that the absolute value function is non-differentiable at $y = \hat{y}$, which leads to undefined gradients at these points. MAE also has slow convergence compared to MSE because of the constant gradients. At the same time, MAE is a good choice if you have outliers because MSE is too sensitive to it.

2.3 Problem 1c

1. **Initialize weights**. The first step is to initialize weights w_0 and w_1 with random values or with small values.
2. **Calculate predictions**. The second step can be achieved by following the formula:

   $$\hat{y} = w_1 x + w_0 \tag{46}$$

   Also, we need to remember to do it for each data point, meaning for w_0^i and w_1^i because otherwise, our predictions will not update.
3. **Define Loss function**. The third step is to choose the loss function which is more suitable for problem-solving.
4. **Calculate Loss function**. The fourth step is to calculate the chosen Loss function for each data point.
5. **Calculate Gradient of the Loss function**. The fifth step is to compute gradients for $\frac{\partial L}{\partial w_0}$ and $\frac{\partial L}{\partial w_1}$
6. **Update of weights**. The sixth step is to update our weights for the next data point. It can be achieved by following the formulas:

   $$w_0^{(i+1)} = w_0^{(i)} - \alpha\frac{\partial L}{\partial w_0}, \quad w_1^{(i+1)} = w_1^{(i)} - \alpha\frac{\partial L}{\partial w_1} \tag{47}$$

   where α is the learning rate, a hyper-parameter that controls the step size during updates.
7. **Repeat of steps**. The last step is to repeat steps 2-6 until we achieve convergence criteria.

2.4 Problem 1d

This is an approximate iterative learning process by gradient descent on the loss surface:

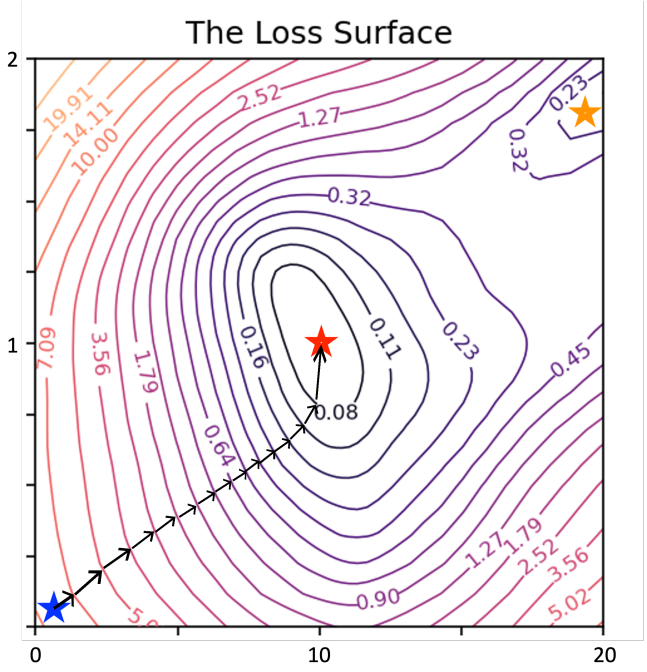

Figure 1: Learning process for Problem 1d

2.5 Problem 1e

The formula for updating predictions is:

$$\hat{y} = w_1 x + w_0 \tag{48}$$

For this problem, the MSE loss function was used:

$$L_{\text{MSE}}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{49}$$

The gradients of the MSE loss function are:

$$\frac{\partial L_{\text{MSE}}}{\partial w_0} = -\frac{2}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i) \tag{50}$$

$$\frac{\partial L_{\text{MSE}}}{\partial w_1} = -\frac{2}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i) x_i \tag{51}$$

And weights are updated by following formulas:

$$w_0^{(i+1)} = w_0^{(i)} - \alpha \frac{\partial L}{\partial w_0}, \quad w_1^{(i+1)} = w_1^{(i)} - \alpha \frac{\partial L}{\partial w_1} \tag{52}$$

In gradient descent, each update to the weights w_0 and w_1 is determined by the gradients, directing the process along the path of steepest descent on the loss function. As the weights get closer to the minimum, the gradient's size diminishes, causing the updates to become smaller until convergence is reached.

2.6 Problem 1f

This is an approximate iterative learning process by gradient descent on the loss surface with a large learning rate:

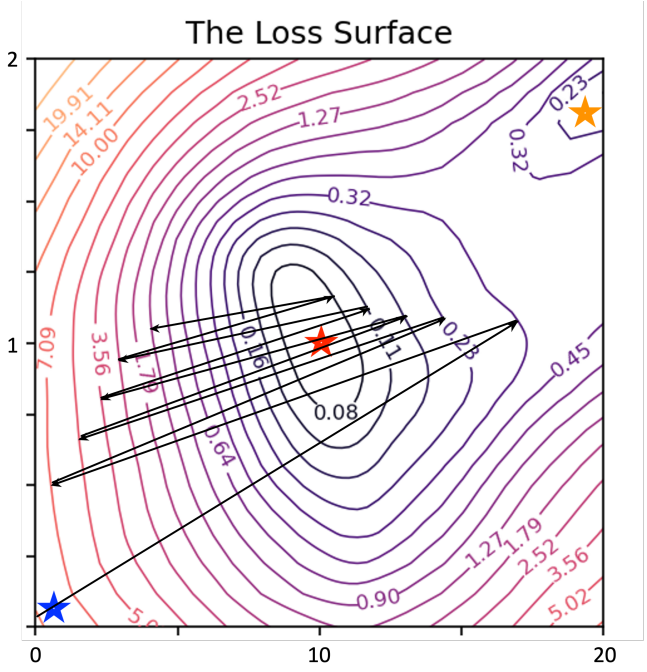

Figure 2: Learning process for Problem 1f

Every gradient is connected with a weight perpendicularly. This approximation is not perfect since I tried to achieve 90 degrees with every step.

2.7 Problem 1g

Several strategies can be used to solve the problem where weights may get stuck in a local minimum:

Optimize learning rate

If the learning rate is too large, the optimization process can oscillate around minima and take more time to converge. MAE with a large learning rate might even not reach the convergence but with a small learning rate, it will take a small amount of time to reach it. For MSE it is a bit different since a large learning rate makes the function use more time for convergence and a very small learning rate also causes a big usage of time. The main idea of this strategy is to try to optimize the learning rate based on your loss function, so it will take less time and stop oscillating around minima.

Random Restarts

Running gradient descent from different random starting points (weights) on the loss surface increases the likelihood of finding the global minimum. Since each run may land in a different local minimum, repeating the process multiple times raises the chances that at least one run will reach the global minimum.

Momentum term

Instead of simply updating weights based on the current gradient, we can use momentum (meaning giving speed in the direction of the weights) in the updates of the gradient to potentially make the weights move past shallow local minima. This strategy allows the optimizer to maintain a consistent direction of movement. The inspiration for this strategy was obtained from lecture notes.

Severals improvement for the SGD

Example: heavy ball

$$w_{t+1} = w_t - \alpha \nabla E(w_t) + \beta(w_t - w_{t-1})$$

Momentum

$(w_t - w_{t-1})$ is the variation at the previous step

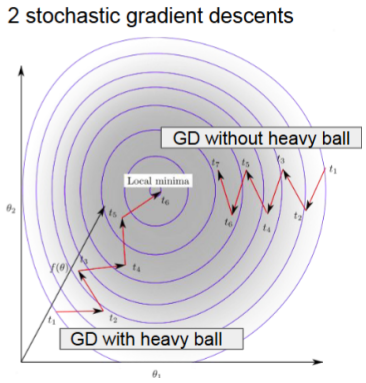

Figure 3: Introduction of Momentum from lecture notes

3 Problem 2

For the second part of this assignment, we will work with CSV file, and two distributions (Gamma and Gaussian), and will explore Bayes' classifier.

3.1 Problem 2a

To load the data I used the same procedure as in assignment 1 - using the Pandas package. When I tried to print the data, I found out that it was hard to read, and therefore transposing of data was used. General information can be obtained through head/tail (shows first rows/last rows), info (shows memory usage, how many rows, how many columns), and describe (count a number of non-null values, count mean, show min or max value).

	0	1
0	8.903629	1.0
1	9.946774	1.0
2	14.458392	1.0
3	9.664572	1.0
4	14.412270	1.0

(a) First rows (data.head)

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3600 entries, 0 to 3599
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       3600 non-null   float64
 1   1       3600 non-null   float64
dtypes: float64(2)
memory usage: 56.4 KB
None
```

(b) Info about data (data.info)

	0	1
count	3600.000000	3600.000000
mean	9.118580	0.555556
std	5.747397	0.496973
min	-3.310259	0.000000
25%	3.710522	0.000000
50%	9.091724	1.000000
75%	13.729211	1.000000
max	25.673673	1.000000

(c) Description (data.describe)

Figure 4: General information

For the variables of histogram I chose the first column of the data (I called it Feature) to be an x-variable and density will be the amount of time the class appears for the Feature value. Histograms look as follows:

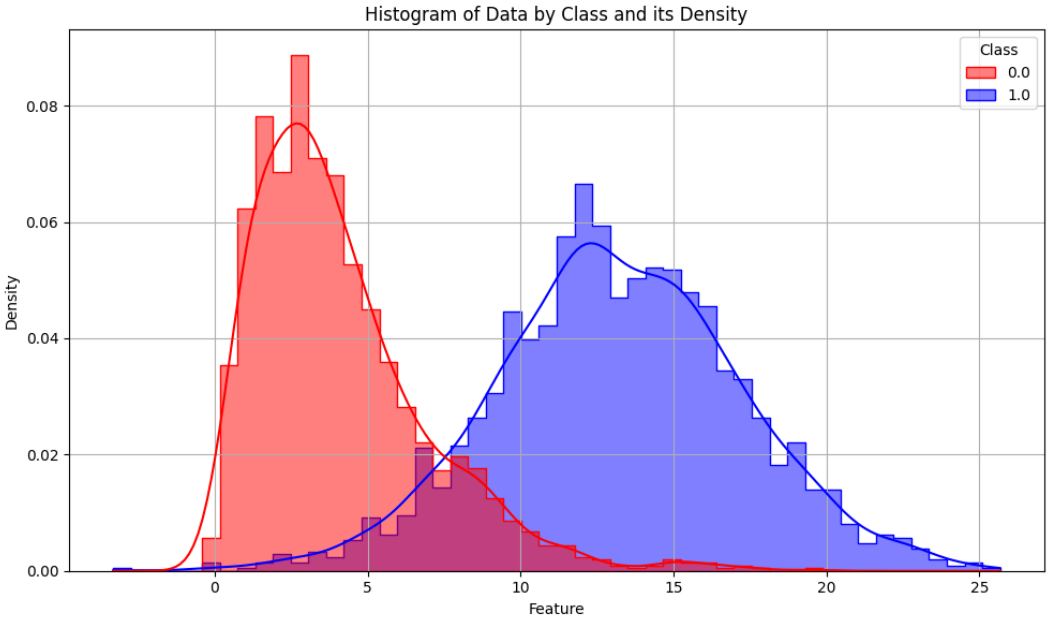

Figure 5: Histogram

As we can see values of Feature for Class 0 are mostly presented on the left (lower values) and values of Feature for Class 1 are mostly presented on the right (higher values). I also checked how many samples have each class: Class 0 has 1600 samples and Class 1 has 2000 samples, which is correct according to the description of the data (it should be 3600 samples).

3.2 Problem 2b

We assume that data from class C_0 follows a Gamma distribution:

$$p(x|C_0) = \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}}, \quad x > 0 \tag{53}$$

And data from class C_1 follows Gaussian distribution:

$$p(x|C_1) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{54}$$

Likelihood function can be represented as:

$$L(\theta) = \prod_{i=1}^{n} f(X_i; \theta) \tag{55}$$

Gamma function:

$$\Gamma(n) = (n-1)! \tag{56}$$

Proving the β with Gamma distribution with $\alpha = 2$ and maximum likelihood estimation as follows:

$$p(x|C_0) = \frac{1}{\beta^2 \Gamma(2)} x e^{-\frac{x}{\beta}} = \frac{1}{\beta^2} x e^{-\frac{x}{\beta}} \quad (\text{since } \Gamma(2) = 1!). \tag{57}$$

$$L(\beta) = \prod_{j=1}^{n_0} \frac{1}{\beta^2} x_0^j e^{-\frac{x_0^j}{\beta}} \tag{58}$$

$$L(\beta) = \left(\prod_{j=1}^{n_0} \frac{1}{\beta^2}\right) \left(\prod_{j=1}^{n_0} x_0^j\right) \left(\prod_{j=1}^{n_0} e^{-\frac{x_j^0}{\beta}}\right) \tag{59}$$

$$L(\beta) = \frac{1}{\beta^{2n_0}} \prod_{j=1}^{n_0} x_0^j \cdot e^{-\frac{1}{\beta}\sum_{j=1}^{n_0} x_j^0} \tag{60}$$

Do the log-likelihood function for Gamma Distribution:

$$\log L(\beta) = \log\left(\frac{1}{\beta^{2n_0}} \prod_{j=1}^{n_0} x_0^j \cdot e^{-\frac{1}{\beta}\sum_{j=1}^{n_0} x_j^0}\right) \tag{61}$$

$$\tag{62}$$

$$\log L(\beta) = -2n_0 \log \beta + \sum_{j=1}^{n_0} \log x_0^j - \frac{1}{\beta}\sum_{j=1}^{n_0} x_0^j \tag{63}$$

To find the Maximization of the Log-Likelihood of β, we are taking take the derivative of the log-likelihood function for β and set whole equation equal to zero:

$$\frac{d}{d\beta} \log L(\beta) = -2n_0 \cdot \frac{1}{\beta} + \frac{1}{\beta^2}\sum_{j=1}^{n_0} x_0^j = 0 \tag{64}$$

Multiplying by β^2 and get:

$$-2n_0\beta + \sum_{j=1}^{n_0} x_0^j = 0 \tag{65}$$

$$\hat{\beta} = \frac{1}{2n_0}\sum_{j=1}^{n_0} x_0^j \tag{66}$$

Since $\alpha = 2$, this simplifies to:

$$\hat{\beta} = \frac{1}{n_0\alpha}\sum_{j=1}^{n_0} x_0^j \tag{67}$$

Proving the σ and μ with Gaussian distribution maximum likelihood estimation as follows:

$$L(\mu, \sigma^2) = \prod_{j=1}^{n_1} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x_1^j - \mu}{\sigma}\right)^2} \tag{68}$$

$$L(\mu, \sigma^2) = \left(\prod_{j=1}^{n} \frac{1}{\sigma\sqrt{2\pi}}\right) \cdot \left(\prod_{j=1}^{n} e^{-\frac{1}{2}\left(\frac{x_1 - \mu}{\sigma}\right)^2}\right) \tag{69}$$

$$L(\mu, \sigma^2) = \frac{1}{(\sigma\sqrt{2\pi})^n} \cdot e^{-\frac{1}{2}\sum_{j=1}^{n}\left(\frac{x_1 - \mu}{\sigma}\right)^2} \tag{70}$$

Do the log-likelihood function for Gaussian Distribution:

$$\log L(\mu, \sigma^2) = \log\left(\left(\frac{1}{\sigma\sqrt{2\pi}}\right)^n \cdot e^{-\frac{1}{2}\sum_{j=1}^{n}\left(\frac{x_1 - \mu}{\sigma}\right)^2}\right) \tag{71}$$

$$\log L(\mu, \sigma^2) = \sum_{j=1}^{n_1} \left(-\log(\sigma\sqrt{2\pi}) - \frac{1}{2} \left(\frac{x_1^j - \mu}{\sigma} \right)^2 \right) \tag{72}$$

$$\log L(\mu, \sigma^2) = -n_1 \log(\sigma\sqrt{2\pi}) - \frac{1}{2\sigma^2} \sum_{j=1}^{n_1} (x_1^j - \mu)^2 \tag{73}$$

To find the Maximization of the Log-Likelihood of μ, we are taking take the derivative of the log-likelihood function for μ and set whole equation equal to zero:

$$\frac{d}{d\mu} \log L(\mu, \sigma^2) = \frac{1}{\sigma^2} \sum_{j=1}^{n_1} (x_1^j - \mu) = 0 \tag{74}$$

$$\hat{\mu} = \frac{1}{n_1} \sum_{j=1}^{n_1} x_1^j \tag{75}$$

To find the Maximization of the Log-Likelihood of σ^2, we are taking take the derivative of the log-likelihood function for σ^2 and set whole equation equal to zero:

9. Derivative of Log-Likelihood for σ^2:

$$\frac{d}{d\sigma^2} \log L(\mu, \sigma) = -\frac{n_1}{2\sigma^2} + \frac{1}{2\sigma^4} \sum_{j=1}^{n_1} (x_j^1 - \mu)^2 = 0 \tag{76}$$

$$-n_1\sigma^2 + \sum_{j=1}^{n_1} (x_j^1 - \mu)^2 = 0 \tag{77}$$

$$\hat{\sigma}^2 = \frac{1}{n_1} \sum_{j=1}^{n_1} (x_1^j - \hat{\mu})^2 \tag{78}$$

And so we get:

$$\hat{\beta} = \frac{1}{n_0\alpha} \sum_{i=1}^{n_0} x_0^j, \tag{79}$$

$$\hat{\mu} = \frac{1}{n_1} \sum_{i=1}^{n_1} x_1^j, \tag{80}$$

$$\hat{\sigma}^2 = \frac{1}{n_1} \sum_{i=1}^{n_1} (x_1^j - \hat{\mu})^2 \tag{81}$$

3.3 Problem 2c

Bayes' Theorem:

$$P(C_k|x) = \frac{P(x|C_k) \cdot P(C_k)}{P(x)} \tag{82}$$

where:

- $P(C_k|x)$ is the posterior probability of class C_k given feature x.
- $P(x|C_k)$ is the likelihood of feature x given class C_k.
- $P(x)$ is the total probability of feature x.

When I implemented the formulas that we were supposed to show in sub-task 2b, I got this result:

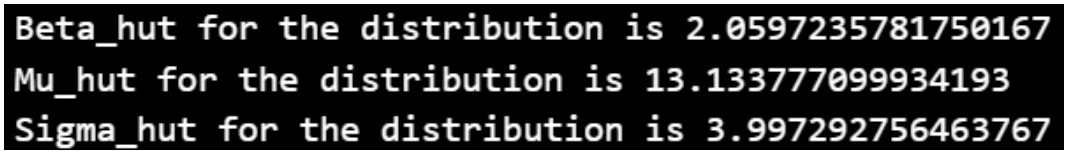

Figure 6: Beta, Mu, and Sigma

By implementing Bayes' Classification in Python, the following accuracy was achieved:

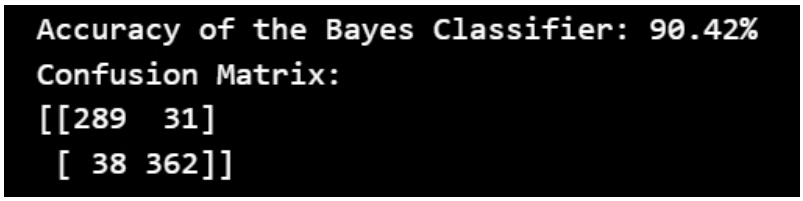

Figure 7: Accuracy

Accuracy is larger than 90 and it means that the classifier gives good accuracy in prediction.

3.4 Problem 2d

The Bayes classifier minimizes misclassification by using Bayes' theorem to choose the class with the highest probability based on the data. It gives the classification a lower error rate because it selects the class that most likely is correct.

The accuracy follows the conclusion (Histogram) of sub-task 2a since I assume that these 9,5 present are those samples that share the same value for Feature, and therefore two on the histogram we see some overlapping of data. Unfortunately, I was not able to complete the plot of misclassification.

4 Conclusion

In this assignment, we work mostly with the theoretical part of the course. We went through common loss functions and saw some of their advantages and disadvantages. In the second part, we used Bayes' Classifier for given data and checked its accuracy (and this accuracy is large). During the process of working on this assignment, I got some help from others and found some inspiration from GeekForGeeks, lecture notes, and other sources that will be presented in the reference list.

5 References

Gribkov, Artemii (2024). *FYS-2021 Machine Learning. Assignment 2.* Github repository. URL (link to the repository).

Ricaud, Benjamin (2024). *FYS-2024 Machine Learning. Lecture Notes.* Notes from

The Pennsylvania State University (2024). *Maximum Likelihood Estimation.* Retrieved from `https://online.stat.psu.edu/stat415/lesson/1/1.2#:~:text=If:%20[u%201%20(x%201,%20x%202,%20%E2%80%A6,%20x%20n)`.

GeeksforGeeks (2024). *Probability Density Estimation - Maximum Likelihood Estimation.* Retrieved from `https://www.geeksforgeeks.org/probability-density-estimation-maximum-likelihood-estimation/`.

GeeksforGeeks (2024). *Bayes Theorem in Machine Learning*. Retrieved from https://www.geeksforgeeks.org/bayes-theorem-in-machine-learning/.

DataCamp (2024). *Naive Bayes Tutorial: Using Naive Bayes Classifier in Scikit-learn*. Retrieved from https://www.datacamp.com/tutorial/naive-bayes-scikit-learn.

Fritz AI (2024). *Best Regression Loss Functions*. Retrieved from https://fritz.ai/best-regression-loss-functions/.

GeeksforGeeks (2024). *Loss Function for Linear Regression*. Retrieved from https://www.geeksforgeeks.org/loss-function-for-linear-regression/#computing.