Assignment 2 FYS-2021

Report by: Sebastian Sjøen-Tollaksvik

All the code is provided on my GitHub page

23.09.24-29.09.24

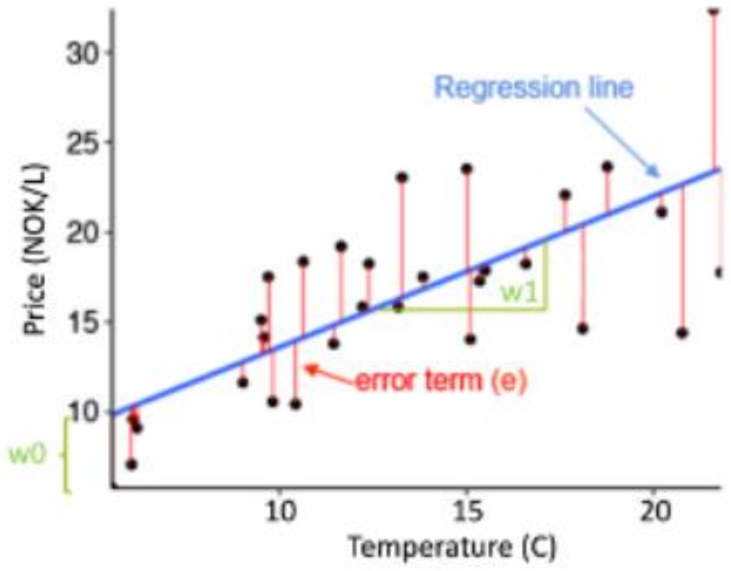
Table of Contents

0.1 Introduction

In the second mandatory assignment in the machine learning course, we will be discussing some fundamental theory about loss functions and gradient decent. Afterward I will be discussing how I made a bayes classifier from scratch by using two statistic distributions.

1.1 choice of loss function

When choosing a loss function for a machine learning model, it is import that you chose a loss function that represents your dataset as best as possible. For example: If there is "outliars" in your dataset, (datapoints that are far away from the rest) it could be smart to use a function that does not punish far away point as much as others. If we square the error then the outliers would have heaps of influence on the dataset.



Since the dataset in the picture provided above seems "balances", I chose to got with a mean square error loss function.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

*N = numbers of features

* Y = result of what the f(x) value provided of a given x

* Y hat = prediction of the model at our current stage.

When trying to adjust the w_1 and w_0 values of our linear regression, we rewrite this function as:

$$E = \sum_i (y_i - (ax_i + b))^2$$

Picture from lecture slide Linear_Logistic_regression.pdf

And we try to adjust a and b, so that the error is as small as possible. We do this by taking the derivative of the function in respect to a and b and use this to adjust the input of the variables w_1 and w_0.

$$\frac{\partial E}{\partial m} = \frac{1}{n} \cdot \sum_{i=0}^{n} 2 \cdot (y_i - (m \cdot x_i + b)) \cdot (-x_i)$$

$$= -\frac{2}{n} \cdot \sum_{i=0}^{n} x_i \, (y_i - (m \cdot x_i + b)$$

$$\frac{\partial E}{\partial b} = -\frac{2}{n} \cdot \sum_{i=0}^{n} (y_i - (m \cdot x_i + b))$$

Screenshot taken from youtube channel NeauralNine

1.2 Gradient descent

When picking loss function, a smooth curve is central. This is because we want to get smaller values out of the loss function. When having a smooth function, it's easy for the model to go in the right direction in the training part. It makes sure that the function converges and that it doesn't get stuck in local minimums. The gradient decent is simple and effective when using the mean square error. This is why it's commonly used in machine learning models. The same cannot be said about the mean absolute error. Since the function outputs an absolute value, it's not continuous around 0. Which means that it's not differentiable around this point. This means that it would need help from numerical methods such as fixed-point iteration or newtons method.

1.3 Adjusting weights using gradient decent

Step 1)

When we adjust the weights using gradient decent, we always starting with giving the weights some staring value(s). If we don't have any information about where the minimum point of the weight(s) is, I usually set the value to zero. Of course, if you have any information that could lead towards a good guess this could speed up the learning process.

Step 2)

Now that we have some weights, we can do our prediction(s) using this formula.

$$\hat{y} = w_1 x + w_0.$$

This formula gives us our prediction, often referred to as y hat using our current weights w_1 and w_0.

Step 3)

Now we need to plot our prediction into the functions given in point 1.1. These functions are specified for using a mean square error loss function, if one wished to use another loss function. You need to derivate in respect of the different weights. These two functions tell us in what direction the weights grow. We can manipulate this information to minimize the loss.

Step 4)

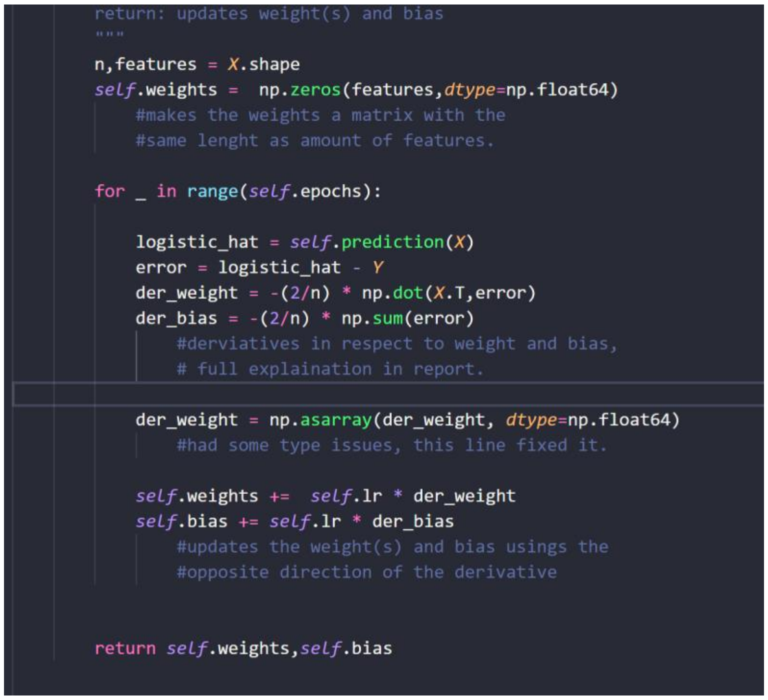
Now adjust the weights in the opposite direction of where the function grows using this formula.

Weight(s) -= learning_rate * der_weight(s)

Where the learning rate is a set number usually between 0.01-0.05

Now we just repeat step 2-4 for a set number of epochs.

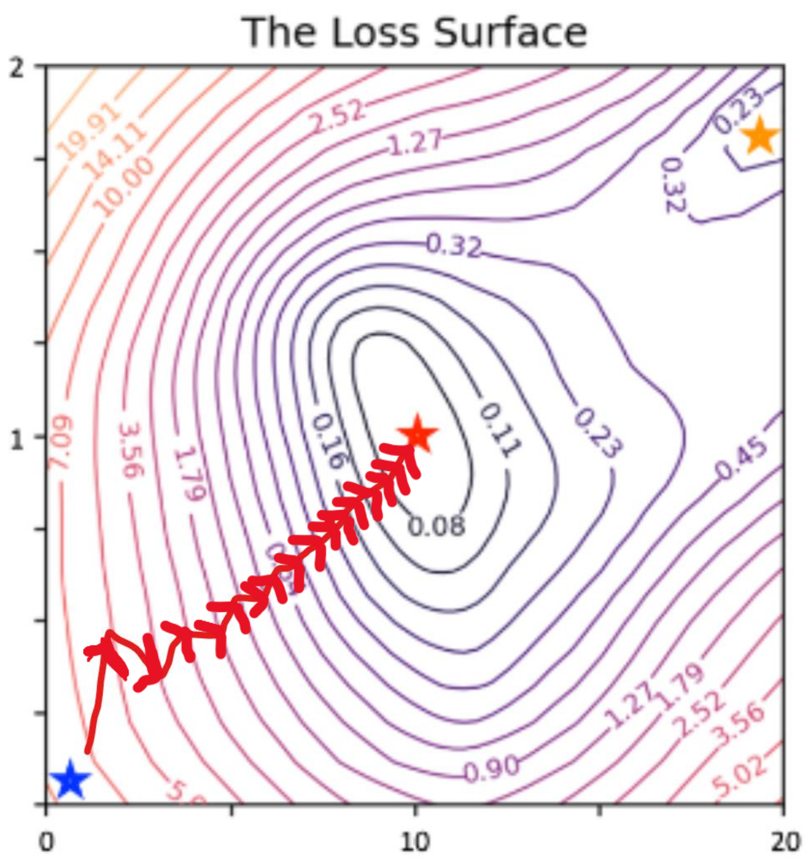
Example:



```
        return: updates weight(s) and bias
        """
        n,features = X.shape
        self.weights =  np.zeros(features,dtype=np.float64)
            #makes the weights a matrix with the
            #same lenght as amount of features.

        for _ in range(self.epochs):

            logistic_hat = self.prediction(X)
            error = logistic_hat - Y
            der_weight = -(2/n) * np.dot(X.T,error)
            der_bias = -(2/n) * np.sum(error)
                #derviatives in respect to weight and bias,
                # full explaination in report.

            der_weight = np.asarray(der_weight, dtype=np.float64)
                #had some type issues, this line fixed it.

            self.weights +=  self.lr * der_weight
            self.bias += self.lr * der_bias
                #updates the weight(s) and bias usings the
                #opposite direction of the derivative


        return self.weights,self.bias
```

Here is an example from the last assignment of how the process looks.

1.4 Gradient decent, small steps



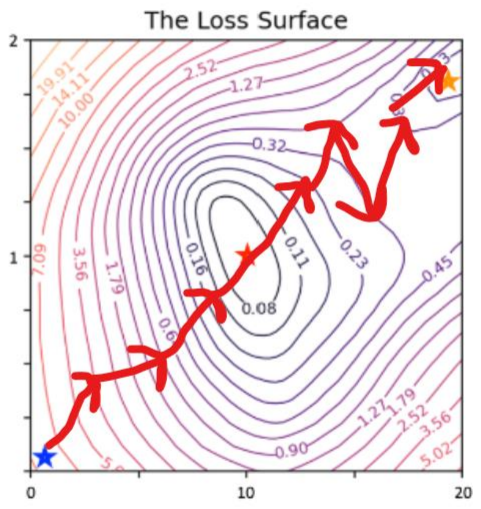
Here is an example of how a gradient decent with small steps could look like, When we are far away from the gradient it takes large steps closer to the minimum point, but when it gets closer, this steps gets smaller and smaller. This is because the grows is smaller at these points.

1.5 Derivatives in gradient decent

When we want to find the smallest possible loss in variables in our machine learning models we use derivatives. Derivatives is a powerful tool which tells us how much a function grows at a particular point. This information can be used to find minimum point. When we use this information, we also want to control the speed of the gradient decent. This is something called **learning rate (LR)**. A small learning rate ensures that we don't skip over the minimum point by going to fast, but we risk getting stuck in local minimum points.

In the context of the plot, the goal is to reach the red star, staring at the blue star. By having a small LR we can convert toward the minimum point slowly but surely. The steps get decreased gradually because of the grows at these points.

1.6 Large learning rate

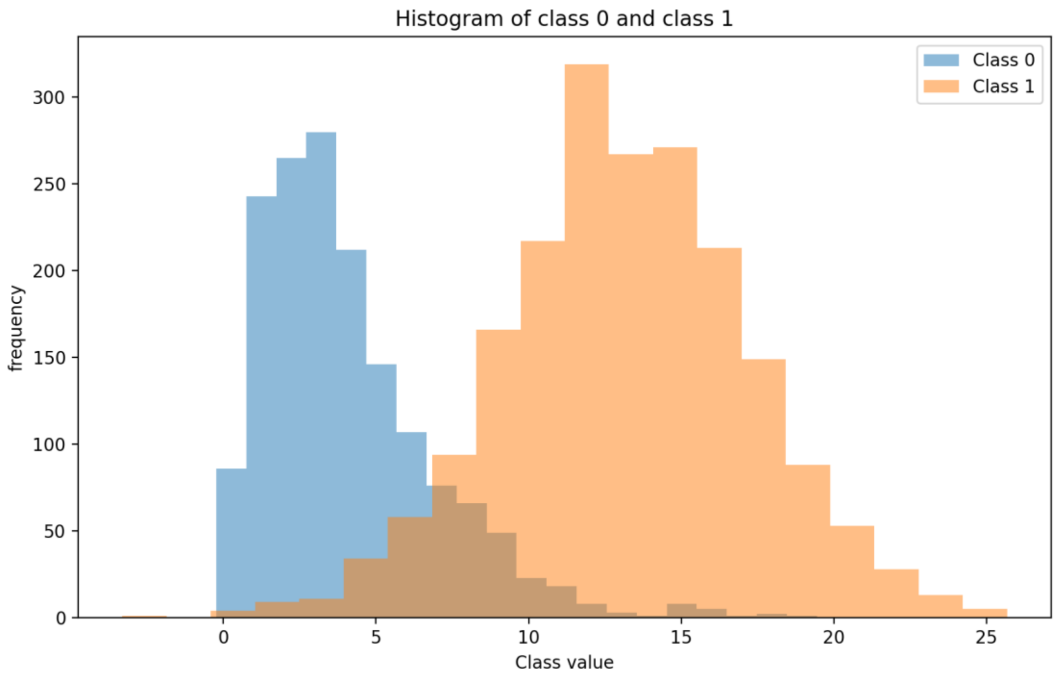


Here the gradient decent overshoots because the LR is too big. It also ends up getting stuck in a local minimum.

Avoid getting stuck in local minimum

During the training process, the gradient decent may get stuck in a local minimum. Especially if we are working in higher dimensions. Sometimes we need to do more than to just adjust the LR. Here we can include momentum. By keeping track of earlier descents, we can avoid getting stuck. Imagine biking down a hill. The momentum from earlier will help you to keep up your pace in the future. This concept can also help in gradient decent.

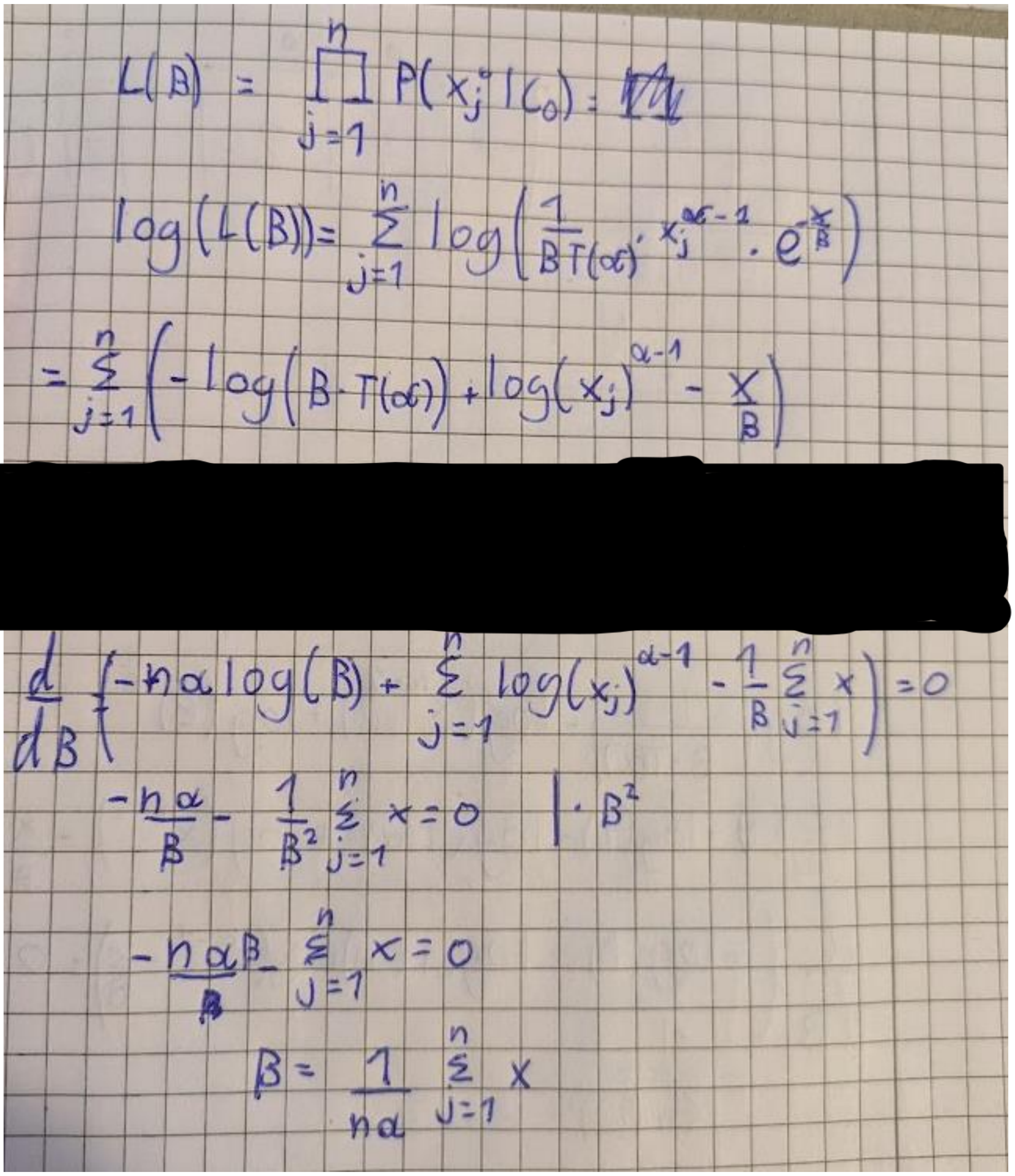
2.1 Loading dataset



In the dataset we have 3600 samples, 1600 of which is class 0 and 2000 is class one. From the looks of this histogram, I conclude that the assumption about the distribution of classes is right in the assignment. I also predict that the classifier will have problem assigning the right values under where the two classes crash. (5-10 range)

2.2 Proof of MLE of parameters

In order to prove that the maximum likelihood estimation of each parameters you just need to derivate the logarithm og the MLE and set it to equal zero:



$$L(B) = \prod_{j=1}^{n} P(x_j \mid C_0) = [illegible]$$

$$\log(L(B)) = \sum_{j=1}^{n} \log\left(\frac{1}{B\Gamma(\alpha)} \cdot x_j^{\alpha-1} \cdot e^{-\frac{x}{B}}\right)$$

$$= \sum_{j=1}^{n} \left(-\log(B \cdot \Gamma(\alpha)) + \log(x_j)^{\alpha-1} - \frac{x}{B}\right)$$

[redacted]

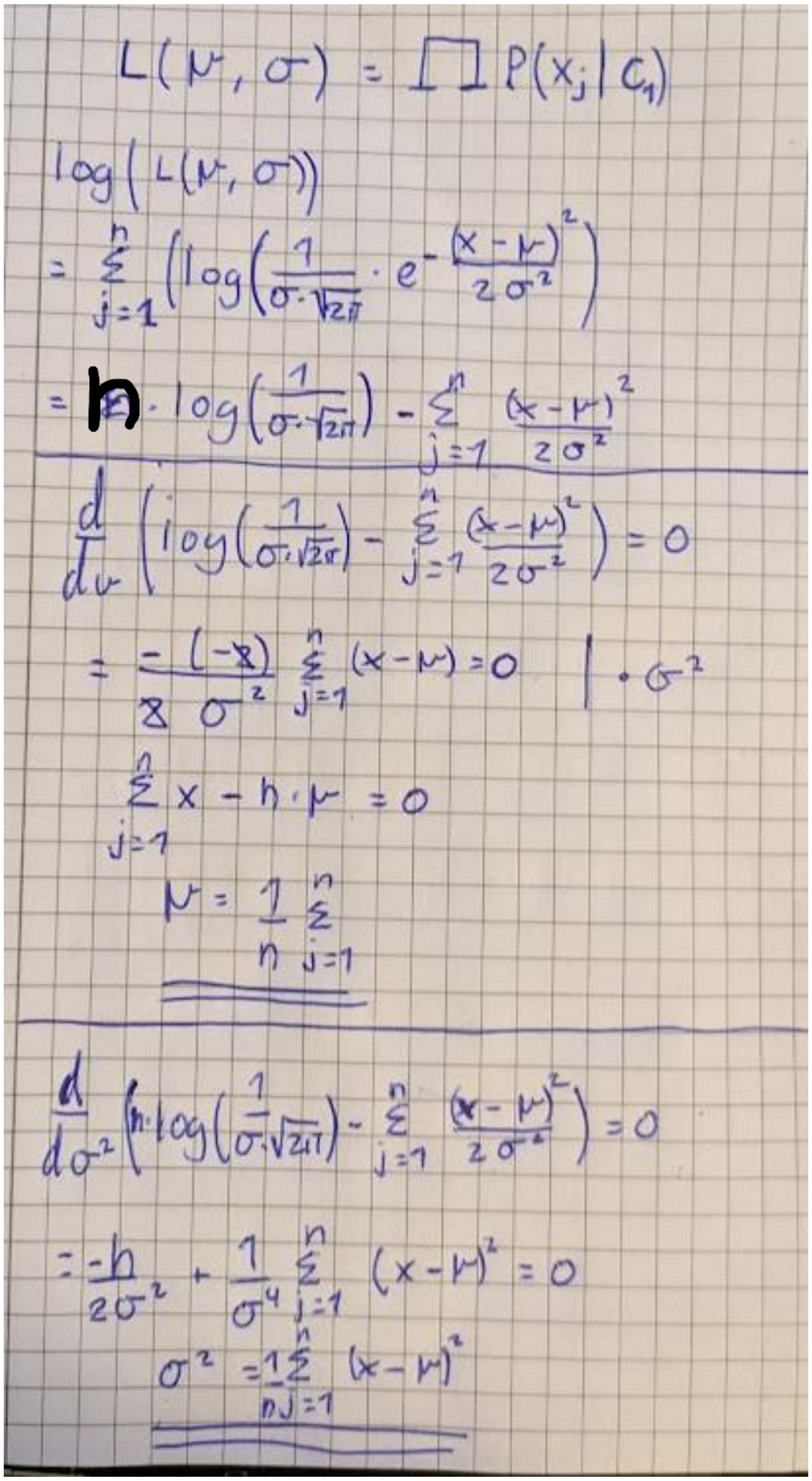
$$\frac{d}{dB}\left(-n\alpha\log(B) + \sum_{j=1}^{n} \log(x_j)^{\alpha-1} - \frac{1}{B}\sum_{j=1}^{n} x\right) = 0$$

$$-\frac{n\alpha}{B} - \frac{1}{B^2}\sum_{j=1}^{n} x = 0 \quad \Big| \cdot B^2$$

$$-\frac{n\alpha B}{\cancel{B}} - \sum_{j=1}^{n} x = 0$$

$$B = \frac{1}{n\alpha}\sum_{j=1}^{n} x$$

Note: The simplification at the start is also the log(MLE) for the distribution



$$L(\mu, \sigma) = \prod P(x_j \mid C_1)$$

$$\log(L(\mu, \sigma))$$

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

$$= n \cdot \log\left(\frac{1}{\sigma \cdot \sqrt{2\pi}}\right) - \sum_{j=1}^{n} \frac{(x-\mu)^2}{2\sigma^2}$$

$$\frac{d}{d\mu}\left(\log\left(\frac{1}{\sigma \cdot \sqrt{2\pi}}\right) - \sum_{j=1}^{n} \frac{(x-\mu)^2}{2\sigma^2}\right) = 0$$

$$= \frac{-(-\cancel{2})}{\cancel{2}\,\sigma^2} \sum_{j=1}^{n} (x-\mu) = 0 \quad \Big| \cdot \sigma^2$$

$$\sum_{j=1}^{n} x - n \cdot \mu = 0$$

$$\mu = \frac{1}{n} \sum_{j=1}^{n}$$

$$\frac{d}{d\sigma^2}\left(n \cdot \log\left(\frac{1}{\sigma \cdot \sqrt{2\pi}}\right) - \sum_{j=1}^{n} \frac{(x-\mu)^2}{2\sigma^2}\right) = 0$$

$$= \frac{-n}{2\sigma^2} + \frac{1}{\sigma^4} \sum_{j=1}^{n} (x-\mu)^2 = 0$$

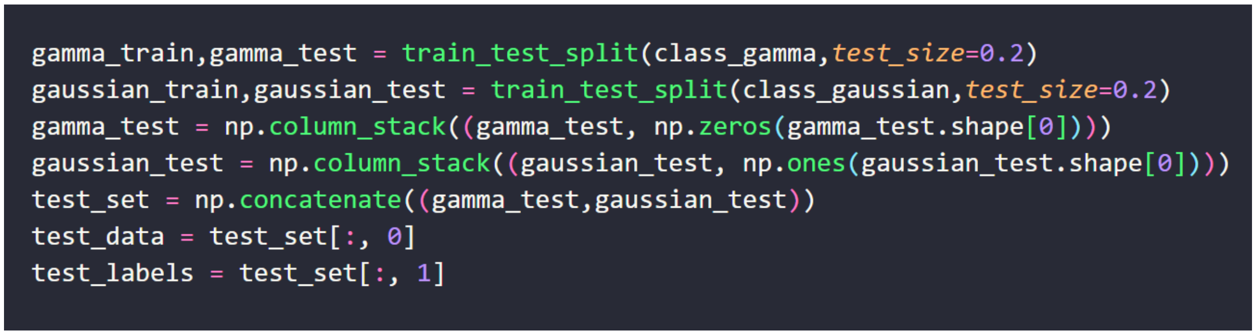
$$\sigma^2 = \frac{1}{n} \sum_{j=1}^{n} (x-\mu)^2$$

Note: The simplification at the start is also the log(MLE) for the distribution

2.3 Building the ML model

To implement the bayes classifier, I split the dataset given and plotted on point 2.1 into test and training using the build in function test_train_split.

Then I added back the labels to the test set so that it would be possible to check the accuracy of the model:



```
gamma_train,gamma_test = train_test_split(class_gamma,test_size=0.2)
gaussian_train,gaussian_test = train_test_split(class_gaussian,test_size=0.2)
gamma_test = np.column_stack((gamma_test, np.zeros(gamma_test.shape[0])))
gaussian_test = np.column_stack((gaussian_test, np.ones(gaussian_test.shape[0])))
test_set = np.concatenate((gamma_test,gaussian_test))
test_data = test_set[:, 0]
test_labels = test_set[:, 1]
```

Afterward I calculated the variance, mean and beta using the formulas provided in the assignment. Using these variables, I was able to build a log likelihood function for each class using the math provided in point 2.2. Since the likelihood that a value is assigned to a class is given as:

$$p(x|C_1)\,p(C_1) > p(x|C_2)\,p(C_2)$$

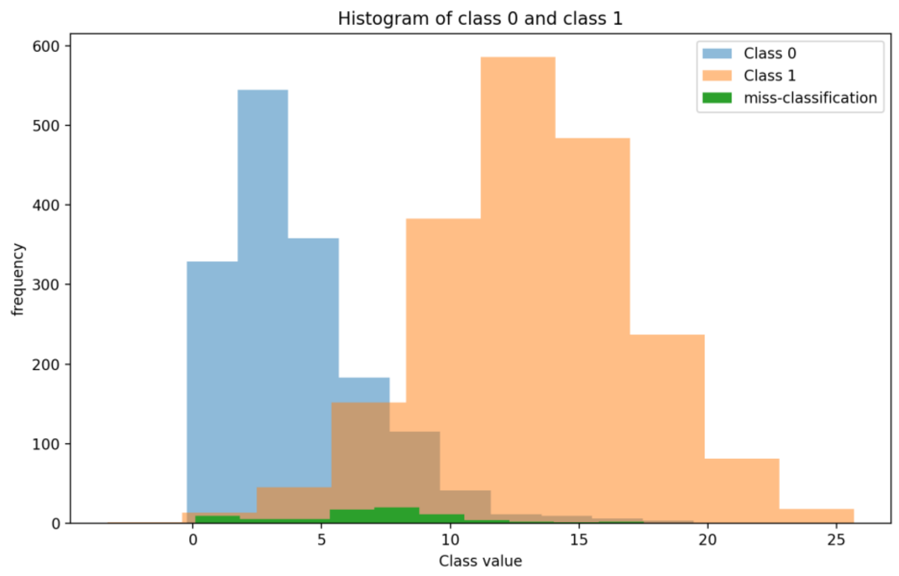
Picture from lecture slide bayes_classification_pt1.pdf

We also need to add the log(prior) to each function. I easily calculated the priors by dividing the amount of each class with the whole dataset.

Afterward I make a simple compare class that checks which of the following classes has the highest values and compared this to the answers in the test set. I ended up with an 90% accuracy which is more than expected.

2.4 Miss-classification

When the probability distribution of the classes is known, we can minimize the likelihood of miscalculation, because we can find out the probability of the classes given the value(s). This method utilizes all the available information about the dataset and therefore minimizes error In this particular model.



I the model had a higher accuracy than expected, but the miss-classifications still mostly happened within that 5-10 range, just as I expected.

3.0 Conclusion

The goals of the assignment were to answer a few questions about loss function and gradient decent. The second part was to build a bayes classifier. The questions is all answered to the best of my knowledge and the accuracy ended up right under the 90% benchmark.

4.0 Sources

Arize. (n.d.). *Mean Square Error (MSE)*. Retrieved from https://arize.com/glossary/mean-square-error-mse/ (25.09.24)

ChatGPT. (09.2024).

NeauralNine . (28.08.2021) *Linear Regression From Scratch in Python*

[Video]. YouTube. https://www.youtube.com/watch?v=VmbA0pi2cRQ (23.09.24)