Assignment No. 2

Loss functions and Classification problems

Thea Gjesdal

October 6th, 2024

Contents

1 Introduction

This report describes the results of the second mandatory assignment of FYS - 2021. The assignment is organized into two main problems with several subsections. Problem one required choosing a loss function for a linear regression that would fit best for optimizing parameters through gradient descent, and elaboration about how the gradient descent works. Problem 2 addressed Bayes classification method, misclassification as well as the maximum likelihood estimation. The solutions are presented in this report.

2 Methods

2.1 Loss function

A loss function measures the accuracy of a linear or logic regression's learning process. It shows how far away the predicted point is from the original point. In machine learning, this information is used to improve the learning method's accuracy.

$$E = \sum_i (y_i - (ax_i + b))^2 \to 0$$

In this project, one of the main loss functions is used when calculating the gradient descent.

2.2 Gradient Descent

Gradient descent is a numerical optimization algorithm. This algorithm is used to optimize the parameters, often weights and biases. By optimizing, it iterates through the function adjusting the parameters in the direction of the negative gradient of the loss function until it reaches minimum value.

$$\omega_{n+1} = \omega_n - \alpha \nabla \mathcal{L}(\omega_n)$$

The value of α is the learning rate of the algorithm. It tells the spacing between every iteration. α´s value determines if the algorithm will reach minimum value of the parameters or get lost "jumping" too far for each iteration or being to slow so that it takes ages to reach minimum.

In this case, gradient descent is used to find the global minimum of the loss function provided as a 2D picture of a loss function.

2.3 Bayes Classification

Bayes classification decides if a value belongs to one class or another and belongs in the world of statistics. In the problem given in this assignment, the Bayes classification model is used to calculate the probability of the samples belonging to either a Gaussian distribution or a Gamma distribution.

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

, where y and X are events and P(X) $\neq$ 0.

y, class variable

X, feature vector

P(y—X) = posterior probability of X

P(X—y) = likelihood probability

P(y) = Prior of y

P(X) = Marginal probability

2.4 Probability density function(PDF)

The probability density function specifies how the probability of the variable is distributed across different values. The calculation of the PDF is used in this assignment to calculate the likelihood in both the gaussian and the gamma distribution.

2.5 Maximum likelihood estimation(MLE)

The maximum likelihood estimation is a tool to find the model that fits our data best and is used to find the expression of the parameters used in problem 2.

The steps in MLE:

1. Fit the distribution to the MLE
2. Take the logarithm of the given function
3. Derivate with regard to chosen parameter
4. Set the result equal to zero
5. Solve for the chosen parameter

3 Solutions of the problems

3.1 Problem 1

3.1.1 Choosing loss function for linear regression

The most common type of loss function when dealing with linear regression is the Mean Squared Error(MSE). Mean Squared Error measures the distance between the original value and the one that is predicted.

$$\mathcal{L}(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2$$

y_i = original value

$\hat{y}_i$ = predicted value

N = N number of features

The mean squared error is easy to use as the calculations are easy to interpret. A downside is that it is sensitive to outliers and can easily be affected by them, resulting in poor performance due to noise. This sensitivity can also be used to detect and penalize large errors. On the other hand the MSE is differentiable which makes it a good choice for optimization problems and does a good job of measuring the model's accuracy.

Derivative of the mean squared error using the chain rule:

Derivative with regard to ω_0:

$$\frac{\partial \mathcal{L}(\omega_0, \omega_1)}{\partial \omega_0} = \frac{\partial}{\partial \omega_0}[\frac{1}{n}(\sum_{i=1}^{N}(\hat{y}_i - y_i)^2)]$$

$$= \frac{1}{n}[\sum_{i=1}^{N} 2(\hat{y}_i - y_i)(\frac{\partial}{\partial \omega_0}(y_i - \hat{y}_i))]$$

Putting in the linear function, $y_i = \omega_0 + \omega_1 x$

$$= \frac{1}{n}[\sum_{i=1}^{N} 2(\hat{y}_i - y_i)(\frac{\partial}{\partial \omega_0}(\omega_0 + \omega_1 x_i - y_i))]$$

$$= \frac{1}{n}[\sum_{i=1}^{N} 2(\hat{y}_i - y_i)(1 + 0 - 0)]$$

$$= \frac{1}{n}[\sum_{i=1}^{N} 2(\hat{y}_i - y_i)]$$

$$\frac{\partial \mathcal{L}(\omega_0, \omega_1)}{\partial \omega_0} = \frac{2}{n}[\sum_{i=1}^{N}(\hat{y}_i - y_i)]$$

Derivative of ω_1:

$$\frac{\partial \mathcal{L}(\omega_0, \omega_1)}{\partial \omega_1} = \frac{\partial}{\partial \omega_1}[\frac{1}{n}(\sum_{i=1}^{N}(\hat{y}_i - y_i)^2)]$$

$$= \frac{1}{n}[\sum_{i=1}^{N} 2(\hat{y}_i - y_i)(\frac{\partial}{\partial \omega_1}(y_i - \hat{y}_i))]$$

Putting in the linear function, $y_i = \omega_0 + \omega_1 x$

$$= \frac{1}{n}[\sum_{i=1}^{N} 2(\hat{y}_i - y_i)(\frac{\partial}{\partial \omega_1}(\omega_0 + \omega_1 x_i - y_i))]$$

$$= \frac{1}{n}[\sum_{i=1}^{N} 2(\hat{y}_i - y_i)(0 + 1x_i - 0)]$$

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

$$\frac{\partial \mathcal{L}(\omega_0, \omega_1)}{\partial \omega_1} = \frac{2}{n}[\sum_{i=1}^{N} (\hat{y}_i - y_i)x_i]$$

3.1.2 Smooth loss functions and gradient descent

Gradient descent: is best used if we have a smooth loss function as grounding. A smooth loss function can for example be the mean squared error(MSE) or cross-entropy loss. The reason why the function should be smooth is because they can provide a more stable and efficient optimization process. One of these factors is that they are continuous and have a differentiable curve that makes it easier to compute the gradient and update the parameters without a large chance for oscillation and divergence during the optimization process. The smooth functions also allow a faster convergence when combined with gradient descent. It makes the gradient descent able to take larger steps for each iteration toward the parameter minimum value.

When it comes to using mean absolute error(MAE) as a loss function there are some problems. First it is important to say that MAE is not a smooth function, which makes it non-differentiable at zero, undefined. This can make the parameter updates unstable and give convergence issues because the gradient is not continuous. This makes the MAE function difficult to navigate and raises the problem of getting stuck in a local minimum. Another problem can be too slow convergence because the gradient in these cases can point in different directions and not necessarily to the minimum. The variated direction of the gradient makes it difficult to make progress in these regions. One last point on issues that can arise when using MAE is that they are generally sensitive to outliers. The outliers can dominate the loss function which can make the gradient point in the wrong direction. This can also result in divergence or convergence to a suboptimal solution.

3.1.3 Optimizing of weights by using gradient descent

The weights in linear regression are the function's gradient and the number of values is represented by the input size of the model. Number of features = number of weights. The goal is to optimize the model so that the loss is as small as possible. As mentioned gradient descent is an algorithm that minimizes the weights by iteratively adjusting them. As shown in the figure each iteration brings the parameter closer to minimum. Pointing in the direction of the steepest decrease in the loss function. The iteration ensures gradient convergence towards the optimal parameter values which leads to the lowest possible loss. It is the partial derivatives for the weights and a learning rate, α on the formula for gradient descent that ensures this behavior. The gradient descent does the same for the bias values.

Gradient descent algorithm:

$$\omega_{n+1} = \omega_n - \alpha \nabla \mathcal{L}(\omega_n)$$

$\mathcal{L}$ = loss function
∇ = gradient sign for the derivative of $\mathcal{L}$
ω = weight - can be a list of values

α = learning rate, large number equals large step, small number equals small steps of iteration, decides convergence speed.

Iteration until minimum is reached: $|\omega_{n+1} - \omega_n| \leq \epsilon$, for a given small ϵ

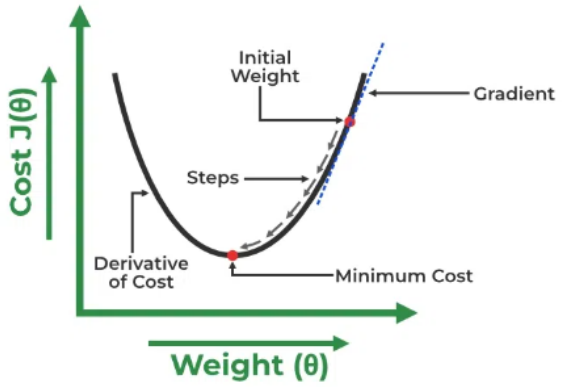

Figure 1: Visualizing Gradient Descent

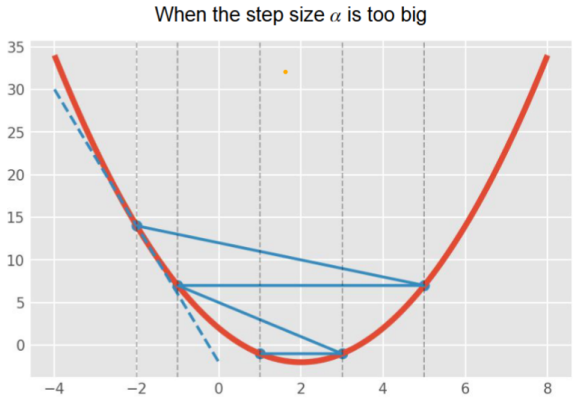

Figure 2: Large learning rate during gradient descent, example 0.1 or 1

Step 1 Initialize weights and bias, setting the initial values randomly or using a specific initialization method. It is possible to use a list of zeroes so that is possible to add new values during iteration.

Step 2 Prediction fase, outputs are calculated using the regression model using the given weights and input data. The result is the predicted values from the training dataset, y_{pred}.

Step 3 Loss calculation, using a loss function such as MSE the difference between the predicted output and the actual output is calculated and printed.

Step 4 Gradient calculation, using the gradient descent algorithm gradients are calculated. Depending on α convergence towards the minimum is fast, slow, or does not get to the minimum point at all. Choosing the learning rate is crucial for this step. If it is too small the step will be so small that it will take a lot of time and it also increases the possibility for the iteration to choose a local minimum instead of the global one. On the other hand, a large learning rate will have too large steps causing the gradients to jump back and forth in the loss function as shown in figure 2. Choosing the optimal step size is possible using the following equation: $\alpha_{optimal} = argmin_{\alpha \geq 0}(\omega_n - \alpha \nabla \mathcal{L}(\omega_n))$

Step 5 Update weights. When new weights are calculated, they are added to the old one, creating a new gradient and continuing the iteration. This minimizes the loss function. The output of weights are the lowest values possible in the given model.

3.1.4 Drawing iterative learning process by gradient descent

Figure 3 shows the gradient descent trail as it iterates toward the minimum on the given loss surface from the starting point, blue star, to the global minimum, red star. Pregiven information about the learning rate is that it is sufficiently small.

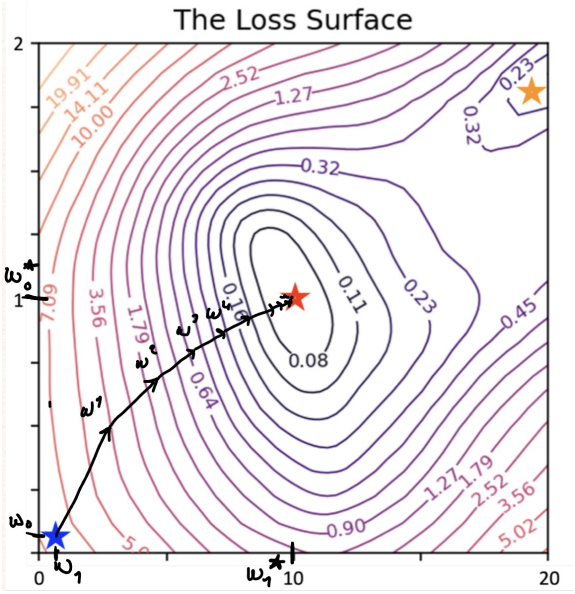

Figure 3: Loss function in the weight space, red star = global minimum, iterative learning process by gradient descent from blue start to red star

3.1.5 Elaborating on the gradient descent figure 3/4

Justifying the direction and shape of the gradient descent in Figure 3. The arrows are getting shorter for every iteration as it comes closer to the global minimum because every new weight value calculated by the gradient descent has a smaller gap between them picturing the steep ascent. The

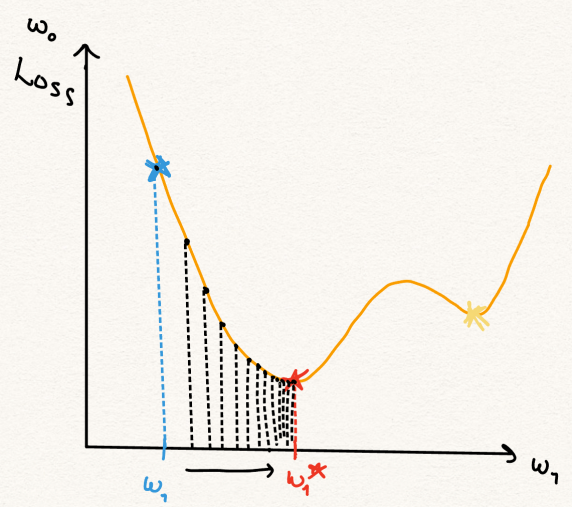

Figure 4: Sufficiently small learning rate

starting direction of the arrow is decided by the first calculations and adjusted for each iteration, here 10.

Assuming a sufficiently small learning rate it is expected that the gradient descent is effective in reaching the minimum not taking any detours on its way or almost stopping due to too small learning rate.

3.1.6 Drawing a trajectory illustrating the iterative learning process - weights fail to converge

Figure 5 shows the gradient descent trail as it iterates 10 times toward the minimum on the given loss surface from the starting point, blue star, toward the global minimum, red star. This time the given value of learning rate is excessively large.

Figure 6 shows the process of large learning rates from the side.

3.1.7 Strategy to solve the problem: getting stuck in local minimum

When handling gradient descent the weights can get stuck in a local minimum. To deal with this problem there is one thing that can be done:

Random restart: Adjust the starting point for weights, by randomizing the starting values of weights it is possible to reduce the chance of ending in a local minimum. It also lets the optimization explore other parts of the weight space by leading to various convergence paths. This gives the gradient descent a large chance of finding the global minimum instead of the local minimum.

Optimze learning rate: Start the gradient descent iteration with a different learning rate. If the optimization process is stuck in a local minimum the reason might be the wrong learning rate for this loss function. Testing for larger and smaller values a more suitable learning rate can be found and the iteration may find the global minimum instead.

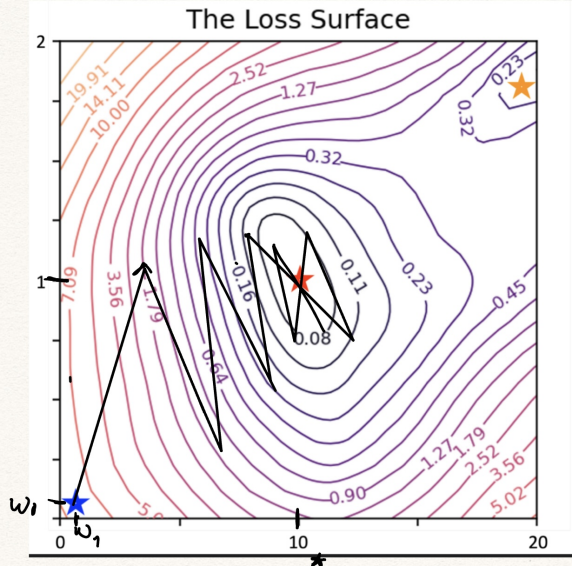

Figure 5: Large learning rate during gradient descent, making the line jump back and forth, example 0.1 or 1

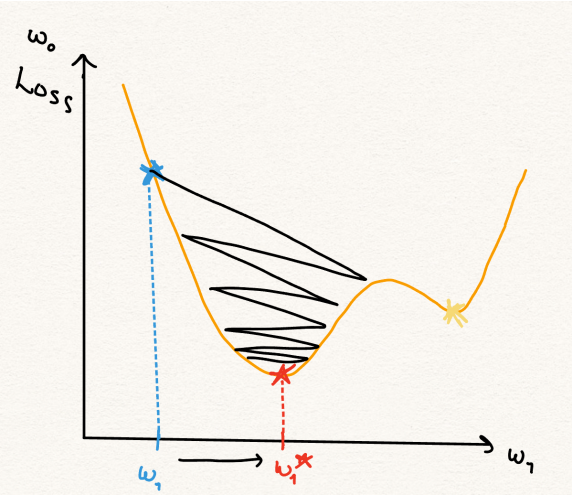

Figure 6: Large learning rate during gradient descent, shown from the side to get a better view

Combining strategies To optimize the learning process it may be necessary to combine the strategies mentioned.

Other possibilities Running smaller batches of data at the same time can help by looking at a smaller part of the weight space at the time. It can give a better understanding of the loss surface.

3.2 Problem 2

3.2.1 Reporting information about *data_problem2.csv* and plotting histogram

From the data set *data_problem2.csv* the output is originally a set of two rows with data. One row with varied selection values, and one with once and zeroes. Rearranging this into two columns makes it easier to interpret and Figure 7 shows how the values of ones and zeroes are distributed.

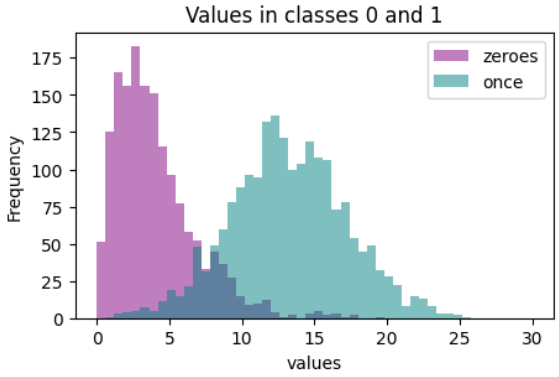

Figure 7: Histogram comparing class zero and one, where zero is Gamma distributed and one is Gaussian distributed

Inspecting Figure 7 the cross-section between class one and class zero is at about value 7. It looks like the cross-section is equally large meaning that it might be hard to separate the values in this area. Where there are only zeroes or ones it should be easier to determine the class of the values and it is obvious that they belong to separate distributions.

3.2.2 Show the maximum likelihood estimates as specific parameters

Assuming that class 0, $\mathcal{C}_0$m follows a Gamma distribution and class 1,$\mathcal{C}_1$, follows a Gaussian distribution the parameters are α and β, $\alpha = 2$, $\beta = unknown$ for Gamma distribution and μ and σ, both unknown, for gaussian distribution.

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

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

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

In deriving the parameters fundamental principles of maximum likelihood estimation(MLE) are used. This is a method used for estimating the parameters of a statistical model.

The likelihood function:

1. Assesses the probability of observing the given data across various parameter values.

2. The definition of the likelihood function uses ω as a parameter and independent observations $x_1, x_2, ..., x_n$:
$$\mathcal{L}(\omega) = P(X = x|\omega) = \Pi_{i_1}^{n} f(x_i|\omega)$$
$f(x_i|\omega$ is the probability density function(PDF) for continuous data.

Log-likelihood:

1. The log-likelihood is often easier to maximize and is therefore widely used,. It is defined as:
$$log\mathcal{L}(\omega) = \sum_{i=1}^{n} logf(x_i|\omega)$$

2. The logarithm transforms products into sums. This simplifies the calculations, especially when dealing with large datasets.

Parameter estimation:

The goal is to find the value of ω that maximizes the log-likelihood function. By deriving the log-likelihood, setting it to zero and then solve for the wanted parameter. The parameter is estimated.
$$\frac{\partial logL}{\partial \omega} = 0$$

Deriving estimations for the parameters, $\hat{\beta}$, $\hat{\mu}$ and $\hat{\sigma}^2$ using maximum likelihood:
Estimations for $\hat{\sigma}$:

$$\mathcal{L}(\omega) = P(X = x|\omega) = \Pi_{i_1}^{n} f(x_i|\omega)$$

As the gaussian distribution is:

$$f(x_1|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Fitting it to the likelihood estimation:

$$L = \Pi_{i_1}^{n} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Now, transferring it to the log-likelihood the result is:

$$logL(\mu, \sigma) = -\frac{n}{2} log(2\pi) - nlog(\sigma) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_1 - \mu)^2$$

Taking the derivative of the equation, sigma:

$$\frac{\partial logL}{\partial \sigma} = -\frac{n}{\sigma} + \frac{1}{\sigma^3} \sum_{i=1}^{n} (x_1 - \mu)^2$$

Next, step is putting the equation equal to zero:

$$-\frac{n}{\sigma} + \frac{1}{\sigma^3}\sum_{i=1}^{n}(x_1 - \mu)^2 = 0$$

Solving for σ^2 gives:

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

Now, deriving the maximum likelihood estimation for μ. The calculations are equal to σ^2 until the partial derivatives:

$$logL(\mu, \sigma) = -\frac{n}{2}log(2\pi) - nlog(\sigma) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_1 - \mu)^2$$

Taking the derivative of μ we get:

$$\frac{\partial logL}{\partial \mu} = -\frac{\partial}{\partial \mu}(\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_1 - \mu)^2)$$

$$\frac{\partial logL}{\partial \mu} = \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_1 - \mu)^2)$$

Setting the equation to zero:

$$\sum_{i=1}^{n}(x_1 - \mu) = 0$$

Simplifying:

$$\sum_{i=1}^{n} x_1 - \sum_{i=1}^{n}\mu) = 0$$

$$\sum_{i=1}^{n} x_1 - n\mu = 0$$

Then the maximum likelihood estimations of the parameter $\hat{\mu}$ equals:

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

For deriving $\hat{\beta}$ we need to start with the Gamma distribution:

$$f(x|\beta, \alpha) = \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}}$$

Adapting it to the maximum likelihood estimation:

$$L = \Pi_{i=1}^{n} \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}}$$

Taking the log-likelihood:

$$logL = -nlog(\Gamma(\alpha) - n\alpha log(\beta) + (\alpha - 1)\sum_{i=1}^{n} log(x_0) - \frac{1}{\beta}\sum_{i=1}^{n} x_0)$$

Derivative based on β:

$$\frac{\partial logL}{\partial \mu} = -n\frac{\partial}{\partial \beta} log(\beta)\alpha + \frac{1}{\beta^2}\sum_{i=1}^{n} x_0$$

$$\frac{\partial logL}{\partial \mu} = -\frac{n\alpha}{\beta} + \frac{1}{\beta^2}\sum_{i=1}^{n} x_0$$

Setting the equation equal 0:

$$-\frac{n\alpha}{\beta} + \frac{1}{\beta^2}\sum_{i=1}^{n} x_0 = 0$$

$$\sum_{i=1}^{n} x_0 = \frac{n\alpha}{\beta} + \frac{1}{\beta^2}\sum_{i=1}^{n} x_0$$

This gives us the maximum likelihood estimation of $\hat{\beta}$:

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

3.2.3 Test accuracy

After splitting the data into training and test data the maximum likelihood estimations are used to estimate the parameters based on the training data. First, it is done for the Gamma distribution and then for the Gaussian distribution. Second, they are put together in a Bayes classification to decide the class of each predicted value in the first step. The accuracy of this implementation had a result of 81%. The classifier managed to sort well for the Gaussian distribution but not so well for the Gamma distribution as can be seen in the print of the Bayes classification.

3.2.4 Minimizing Bayes´s classifier and showing misclassified data

The Bayes classifier minimizes the probability of miss-classification when the probability distribution of the data is known because it maximizes likelihood and prior so that posterior probability gets the highest possible value. Evidence can be ignored because it is a constant and therefore is not important when a function is maximizing. The maximizing gives an optimal decision boundary and gives a high accuracy.

Figure 8 shows vividly how the values in the Gaussian distribution as a lot more correctly classified values than those of the Gamma distribution. It shows that the conclusion in a) is partly correct since it miscalculates a lot in all of the Gamma distribution.

The reason why so many values were misclassified might be a result of the gamma functions layout or the way beta is calculated. It is assumed that α equals 2. This could affect the posterior probability of the Gamma distribution in a negative way if it is not ideal for the PDF. There is also a possibility that the prior can be optimized in a better way.

Calculation of beta:

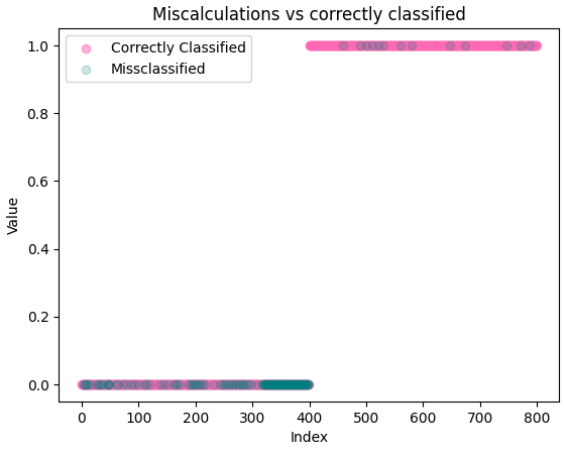

Figure 8: A scattered plot of the miscalculated values in the Bayes classifier, teal color

```
beta = np.array([np.mean(C0_train) / (np.percentile(C0_train, 75) - np.percentile(C0_train, 25))])
```

Looking at the confusion matrix there are not many predictions that are false negative. Most of them are false positives.

```
[[258 142]
 [ 13 387]]
```

4 Conclusion

This report presents the solutions for the second assignment. The main topics brought up and how to use them are, loss functions, gradient descent, bayes classification, and maximum likelihood estimation. When using gradient descent the loss function should be Mean squared error and the learning rate has to have a value that is not too large or too small to reach the minimum. Furthermore, it is shown that the Bayes classifier has an accuracy equal to 80%.

References

[1] GeeksForGeeks, https://www.geeksforgeeks.org/ml-linear-regression/*Linear Regression*

[2] GeeksForGeeks, https://www.geeksforgeeks.org/loss-functions-in-deep-learning/*Loss function*

[3] Arizone.edu, https://math.arizona.edu/ jwatkins/o-mle.pdf *Topic 15: Maximum Likelihood Estimation*

[4] Statology, https://www.statology.org/misclassification-rate/ *Misclassification Rate in Machine Learning: Definition & Example*