Scientific Report for course FYS-STK4155

Project n°3

Analyzing Wine Quality Dataset as a Classification Problem

Sofie Vos, Kristian Bugge, Mathieu Nguyen

Department of Physics
University of Oslo
Oslo, Norway, 2023

Abstract

In this project, we applied numerous methods in order to predict the value of instances from Scikit-Learn's Digit Dataset. We have used Feed Forward Neural Network, Logistic Regression and Gradient Boosting and tried to optimize their needed parameters (such as learning rate, number of steps, etc...). After optimizing them, we achieved accuracy scores of respectively 98.4%, 97% and 97%. We then compared our results to past studies made on that same dataset and see how they hold, as well as explain the eventual differences. Lastly, we concluded that the Neural Network surpass by far the other two methods. This outcome was expected since it is theoretically a more suitable model for deep learning problems.

Contents

1 Introduction

During the semester, we have explored methods in the Machine Learning perspective to predict values given a certain dataset. We also applied them on educational datasets such as **Franke's Function** or the **Wisconsin Breast Cancer Dataset**. These two datasets have been employed for both regression and binary classification problems, and the methods used on them demonstrated strong performance, as discussed in our first two projects. Therefore we wonder if the methods we learned can maintain their efficacy when applied to the analysis of a more complex dataset, such as a **multiclass classification problem**. That is the challenge we tackle on this project. Hence, we will select a restricted number (but very well performing) methods, which are logistic regression, feed forward neural networks and gradient boosting, and observe how they predict our multiclass dataset, the MNIST Digit dataset. The reason for choosing this dataset is because it is quite small in size, well-structered and easy to use.[**MNIST**]

2 Our Dataset : MNIST Digit Dataset

Our dataset studied in this project will be the MNIST Digit Dataset, a dataset dedicated to handwritten digits, released in 1999. It is an **extremely popular dataset** that has served as the basis for **benchmarking classification algorithms**, even nowadays.
Most importantly, we want to focus on **Scikit-Learn's version of the MNIST Dataset**, as it is much smaller both in sample size and quality image, making it a better fit for this project.
It is composed of **5620 images**, each with **64 features** representing the 64 pixels composing the image (as each instance is an 8x8 pixel image). Each feature is an **integer ranging from 0 to 16**, based on how colored the pixel is. Indeed, the images are white numbers written on a black background, therefore the higher this value is, the more likely that part of the image has been written on.

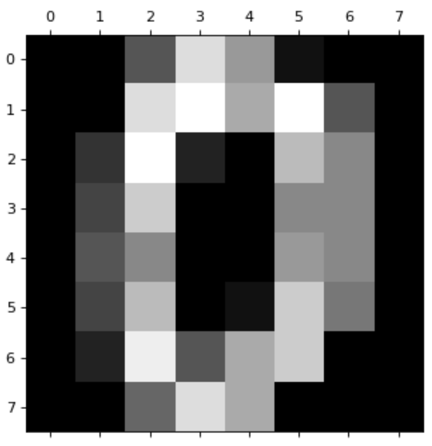


Figure 1: An example of an image present in Scikit-Learn's MNIST Dataset. Since the image is only made of 64 pixel, it can be difficult for a human eye to properly determine its value.

The target column is simply an integer representing the number illustrated in the image. Each class is made of **around 560 images**, making our dataset **very balanced** and reducing the risks of having overrepresented integers.

The quality score is an integer value between 0 and 10, so this dataset can either be interpreted **both as a regression or a classification class**. However for the sake of this project, we will consider this dataset a **multiclass classification problem**.

Our objective here is to create a model that is able to predict what digit the image should illustrate with the best precision as possible. Most importantly we will **reapply the models** that we have already discussed in previous projects and analyze their performance on this new dataset and classification problem (as we have only studied regression and binary classification problems before). Depending on the considered model, we may also fine-tune the hyperparameters used in order to maximize our performances.

2.1 Implemented Methods and Definitions

2.2 Confusion Matrix

A confusion matrix is a performance measurement for machine learning classification problems. We have previously defined this metric in a binary classification problem. However this time we are dealing with a multiclass problem, meaning that our confusion matrix will be a 10x10 matrix.

The values TP, FP, TN and FN are nonetheless still defined the following way :
- True Positive (TP) : A predicted value that is actually in the studied class
- False Positive (FP) : All values predicted in a certain class, excluding the TP
- False Negative (FN) : A predicted value that has been correctly classified out of the studied one
- False Positive (FP) : All values that are actually in a certain class, excluding the TP

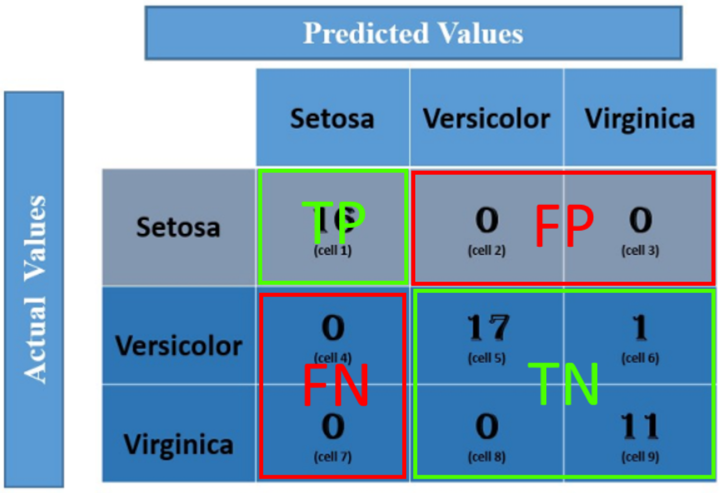


Figure 2: An example of confusion matrix, now applied on a 3-class problem. Although we are currently discussing a 10-class problem, the model applies nonetheless

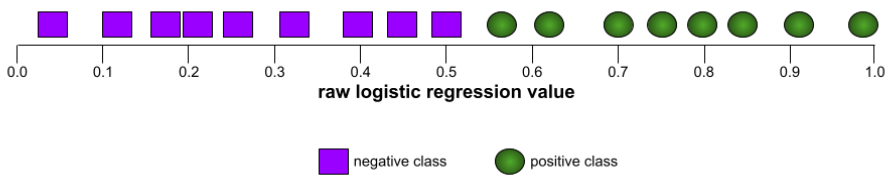
The precision, accuracy and recall metrics are still defined the same way, with more details in our previous report.

2.3 ROC Curve

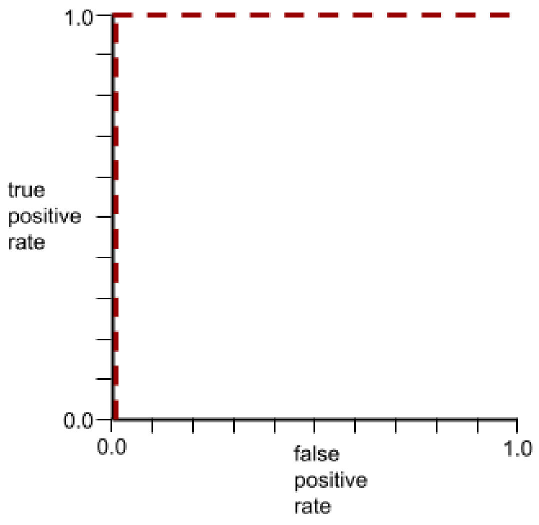
A Receiver Operating Characteristic curve, or ROC curve, is a graphical plot that illustrates the **performance of a binary classifier model** at varying threshold values. [11] It is plotting the **true positive rate (TPR)** against the **false positive rate (FPR)** at each threshold setting, each defined the following way : [5]

$$TPR = \frac{TP}{TP + FN} \qquad\qquad FPR = \frac{FP}{FP + TN} \tag{1}$$

The ROC Curve suggests that our model is **able to separate positive classes from negative classes**, in a binary classification perspective. The curve takes values from 0 to 1, indicating how well it manages to do so (with 1 being the best possible outcome). [6]



(a) Illustration of how a ROC Curve works with raw logistic regression values. In this case, the positive and negative values are perfectly separated, resulting in a perfect ROC Score



(b) Illustration of the ROC Curve depicted by the previous raw values. Here, we can see that the curve instantly bumps to one, as the separation has been done without mistakes.

Figure 3: ROC Curve Illustrations

If the model is completely unable to sort positive and negative classes (meaning that it alternates both values endlessly), then the ROC Curve would represent a **perfectly linear function**.

2.4 Feed Forward Neural Network

The FFNN is the Neural Network algorithm implemented in this project. As a reminder, a FFNN presents itself with **multiple layers of nodes** : an input, hiddens and output layers.

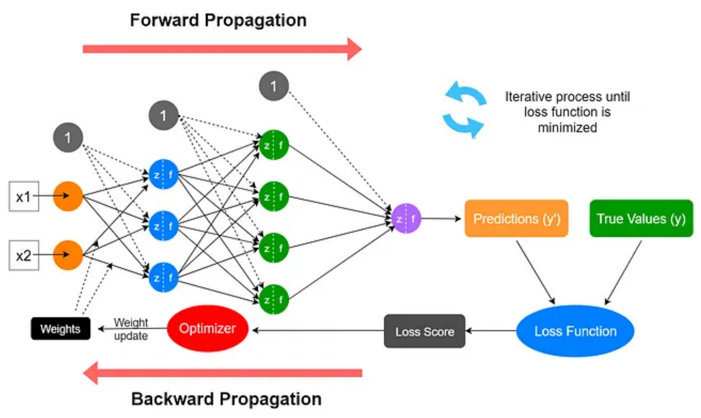


Figure 4: Algorithm and construction of the FFNN. The cost and activations functions can vary depending on if we are discussing a regression or classification problem.

Once again, we are looking to optimize the prediction results of our model by **rigorously selecting our parameters**. This includes the **number of hidden layers**, the **number of nodes per layer**, **which cost function** and **activation function** to select and how to **update the weights of each node**. To understand how to maximize these parameters, we can find more information and details in our **previous project on the matter**.

2.5 Logistic Regression

Logistic Regression is a method which models the probability of a discrete outcome given an input variable, and is also a method that we have already introduced in a former project. However this time, as we are dealing with a multiclass classification problem, we need to adjust this method and **redefine some properties.**

Indeed, we now have a target result in the range of [0,9] rather than the former True/False output, hence each category have a prediction probability of :

$$\begin{aligned} p(G=k|X=x) &= \frac{e^{\beta_{0,k}+\beta_k^T x}}{1+\sum_{k'=1}^{K-1} e^{\beta_{0,k'}+\beta_{k'}^T x}} \\ p(G=K|X=x) &= \frac{1}{1+\sum_{k'=1}^{K-1} e^{\beta_{0,k'}+\beta_{k'}^T x}} \end{aligned} \tag{2}$$

with k = 0,...,K-1 and K = 10
As mentioned in our former project, Logistic Regression tends to **perform less reliably than a Neural Network** in case of large dataset and **numerous possible outcomes**, where a Deep Learning method like the FFNN will

adapt more easily. Therefore Logistic Regression be used as a **stepping stone** to compare whether or not our FFNN actually performs as intended.

2.6 Decision Tree

A decision tree is a **supervised machine learning type** that makes predictions based on how a **previous set of questions or conditional statements were answered**. [3]
An example of a decision tree :

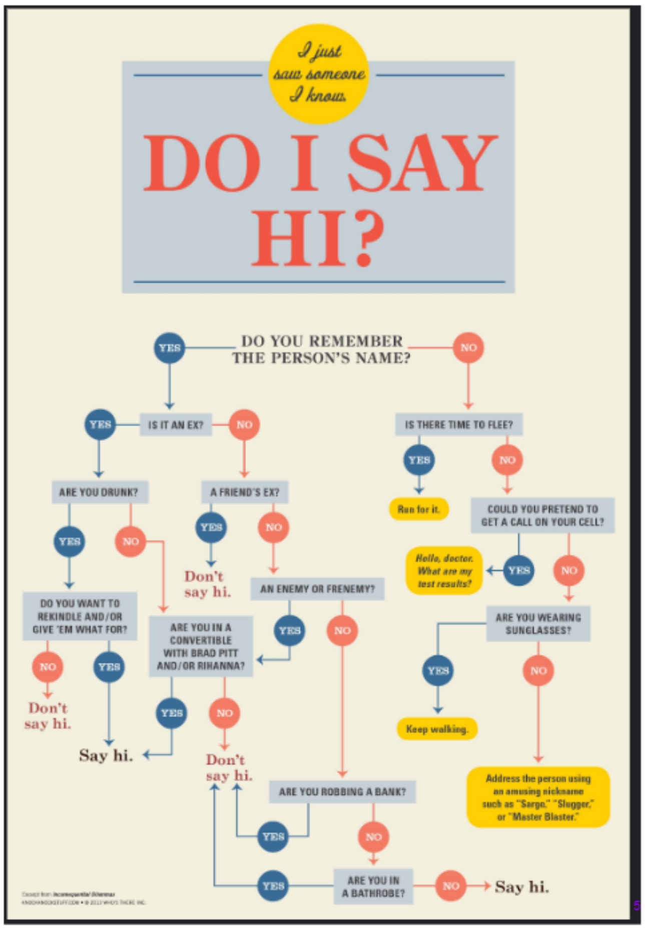

I just saw someone I know.
DO I SAY HI?
DO YOU REMEMBER THE PERSON'S NAME?
YES
NO
IS IT AN EX?
IS THERE TIME TO FLEE?
ARE YOU DRUNK?
A FRIEND'S EX?
Run for it.
COULD YOU PRETEND TO GET A CALL ON YOUR CELL?
Don't say hi.
Hello, doctor. What are my test results?
AN ENEMY OR FRENEMY?
DO YOU WANT TO REKINDLE AND/OR GIVE 'EM WHAT FOR?
ARE YOU IN A CONVERTIBLE WITH BRAD PITT AND/OR RIHANNA?
ARE YOU WEARING SUNGLASSES?
Keep walking.
Say hi.
ARE YOU ROBBING A BANK?
Address the person using an amusing nickname such as "Sarge," "Slugger," or "Master Blaster."
ARE YOU IN A BATHROBE?

Figure 5: Illustration on how a Decision Tree works. [14] Based on multiple conditions and statements, the model will output a value **given the answers of all previous questions**. It is important to note that completely different paths can **still lead to the same output**, just like in an actual decision tree.

Decision trees can be used directly; however, they are likely to suffer from a **high variance** [10]. Therefore, we usually use them as a **backbone for other very performing methods**. Indeed, they are also used in **Random Forest methods** and more importantly here in **Gradient Boosting** to create a composite model.

2.7 Gradient Boosting Classification

Gradient boosting relies on the intuition that the best possible next model, when **combined with previous models**, minimizes the overall prediction error. The idea is to **set the target outcomes** for this next model in order to minimize the error. When combining all the models that each minimize the error, we will get a well performing composite model. [9]

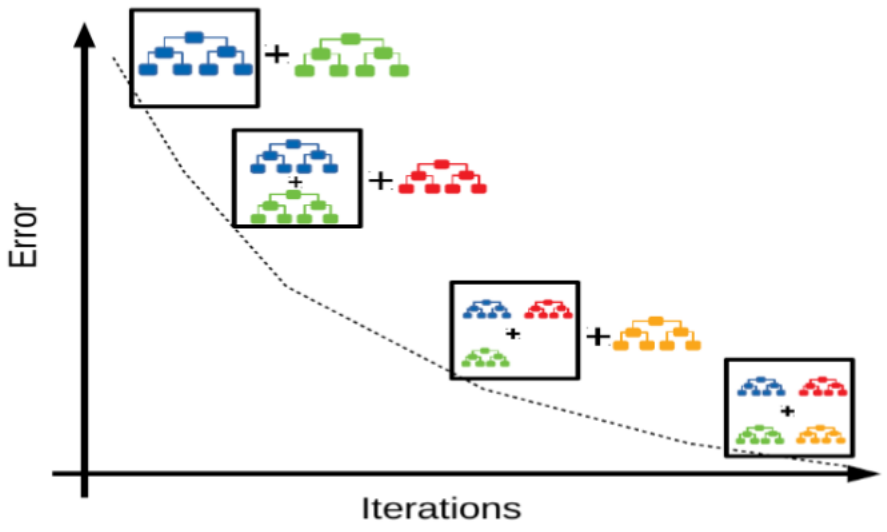


Figure 6: Overall MSE Score of a Gradient Boosting model as the number of boosting iterations increases. [15] By summing all calculations done on each steps, the final model is able to summarize all deductions done to achieve the best possible predictions.

For multiclass classification problems gradient boosting is done by the following steps:
- 1) Set the initial model predictions.
- 2) Repeat the following for each boosting round (the more the better the composite model):
    - Repeat the following for each class:
        - a) Compute the **pseudo residuals**, which are the differences between the actual value and the predicted value. [13]
        - b) **Train a regression decision tree** to predict the pseudo residuals.
        - c) Adjust the tree's predicted values to optimize the objective function.
        - d) Add the new tree to the current composite model. [2]

To show these five steps in more detail, we go through a program implementing the **GradientBoostingClassifier**.

2.7.1 Setting the initial model predictions

Because we classify instances over 10 classes (that are supposedly well balanced), every instance gets a probability of being in a certain class of $\frac{1}{10}$.

The one hot encoder initializes the target values by writing a **0 if the instance is not in the class** and a **1 if it is in the class in y_{ohe}.** The columns of y_{ohe} represent the classes from 0 until (and including) 9 in our case. The rows of y_{ohe} represent the instances.

Then the following steps are done for every class in each boosting round

2.7.2 Compute the pseudo residuals

For all the instances trained on, the pseudo residual is either a 1 if the instance is in the class or 0 if it is not in the class minus the probability of the instance being in the class. The pseudo residuals are the **negative gradients of the Multinomial Negative Log Likelihood.** [2]

$$r_{ik} = -J'(F_k(x_i)) = -[\frac{\delta J(\{y_{il}, F_l(x_i)\}_{l=1}^{K})}{\delta F_k(x_i)}] = y_{ik} - p_k(x_i) \tag{3}$$

2.7.3 Train a regression tree

Here, we simply **fit a decision tree** to the features with the pseudo residuals as target values.

2.7.4 Adjust the tree's predicted values

The leaves in the decision tree might contain multiple pseudo residuals. Therefore, the values in these leaves will be **recomputed to one value** which optimizes the Multinomial Negative Log Likelihood (loss function) for the decision tree. Because taking the derivative of this function is hard, it is **approximated by Taylor** expanding it : [12]

$$L(y_1, F_{m-1}(x_1) + \gamma) \approx L(y_1, F_{m-i}(x_1) + \frac{d}{dF()}(y_1, F_{m-i}(x_1))\gamma + \frac{1}{2}\frac{d^2}{dF()^2}(y_1, F_{m-i}(x_1))\gamma^2 \tag{4}$$

Setting the derivative of the Taylor expansion to 0 and the optimal values for the leafs will be :

$$\gamma = \frac{-\frac{d}{dF()}(y_1, F_{m-1}(x_1)}{\frac{d^2}{dF()^2}(y_1, F_{m-1}(x_1)} \tag{5}$$

2.7.5 Add the new tree

When the training is done, the prediction is found by **going over all models** and **predicting for each class** the probability of the instance being in it. Eventually, the classes that have the highest probabilities are returned.

3 Results and Interpretation

In this section we put into practice the **FFNN**, **Logistic Regression** and the **Gradient Boosting** methods. Namely, the adaptation of the three methods on the Digit Dataset will be observed under different circumstances.

3.1 Feed Forward Neural Network

Our FFNN implementation from project 2 proved quite effective at classifying the Wisconsin breast cancer dataset. So, we will try to optimize it for the MNIST dataset in this project.

3.1.1 Finding the optimal scheduler

First we need to get a picture of how the different schedulers perform on a classification problem like MNIST. The dataset is trimmed to only include the 2's and 5's, to **shorten the running times**, as we are exploring accuracies for both the learning rate and regularization parameter lambda on six different schedulers. Our script "FFNN_MNIST_sched.py" does this, and produces the heatmaps below :

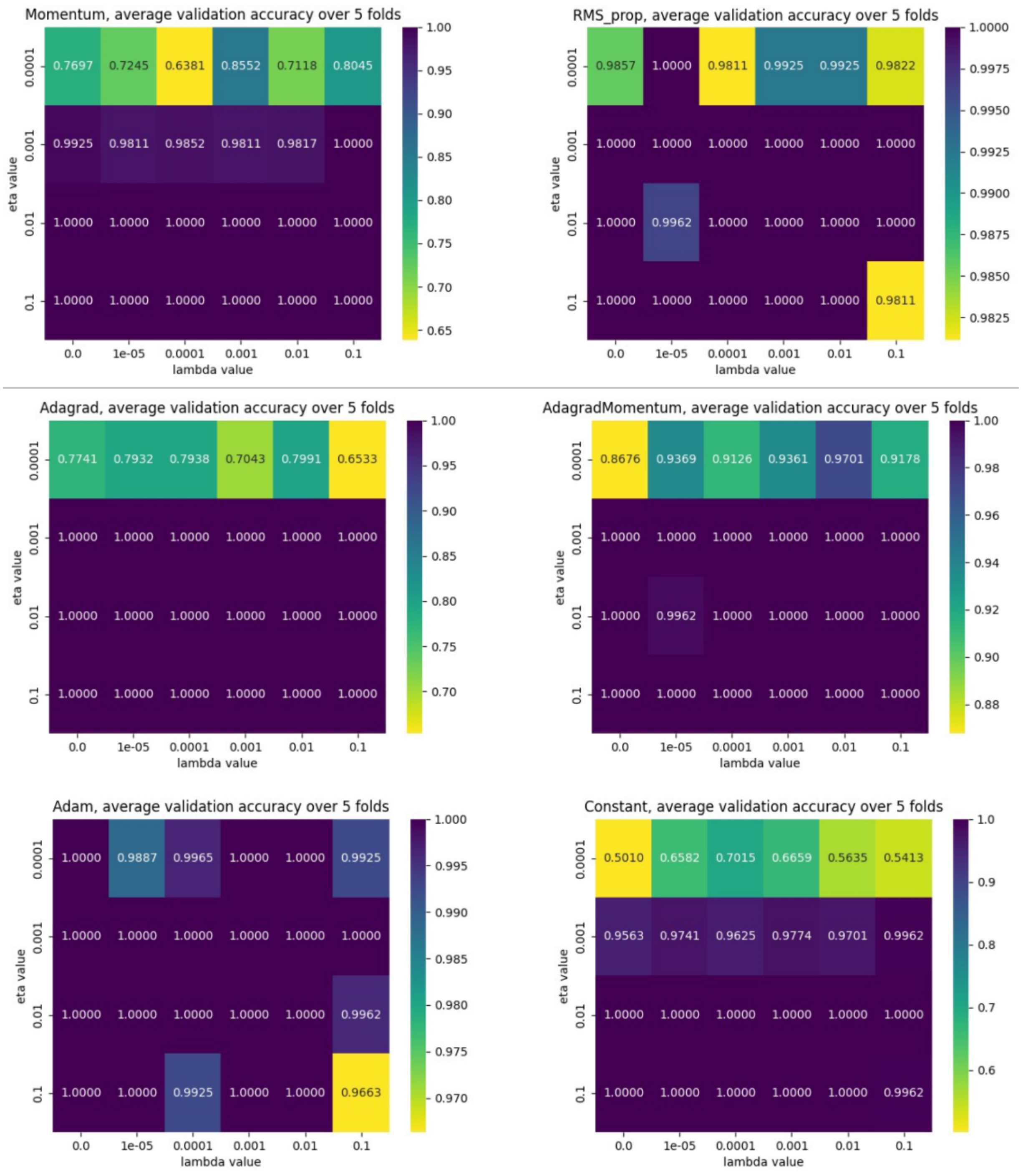


Figure 7: Resulting accuracy of different schedulers used for the FFNN (see names on graph), on MNIST dataset consisting of 2's and 5's, as a function of learning rate η and λ.

All schedulers perform quite well when the dataset is trimmed, for the most part **achieving 100% accuracy**. Still, there are differences in performances and we can see that the Adam scheduler performs well at a **broad range of hyperparameters**. As this is an ideal quality for the case at hand, we choose to proceed with **Adam**.

3.1.2 Analyzing the entire dataset

Now we will do the same thing again for Adam, only now we introduce the entire MNIST dataset (digits 0-9). Doing so in the script "FFNN_MNIST_adam.py", we get the below heatmap as a result:

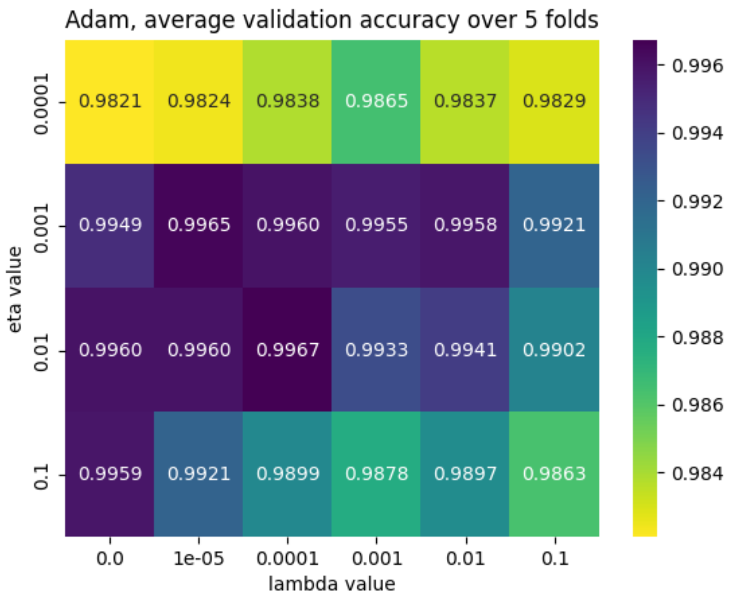
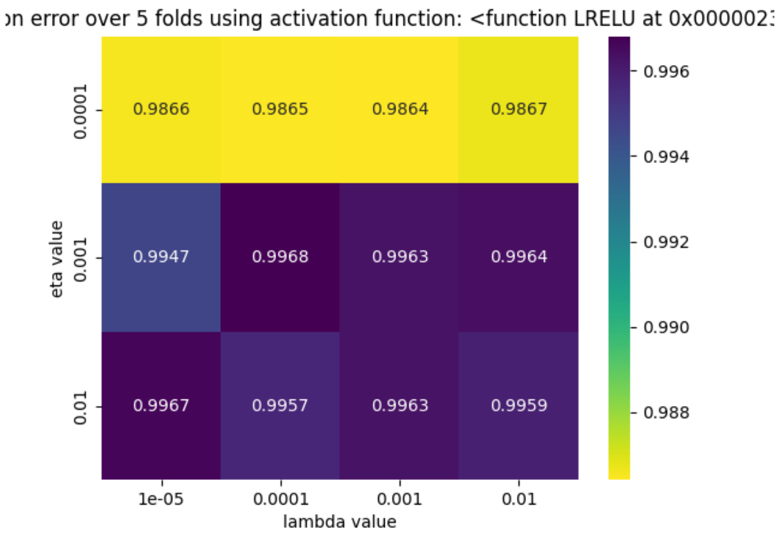


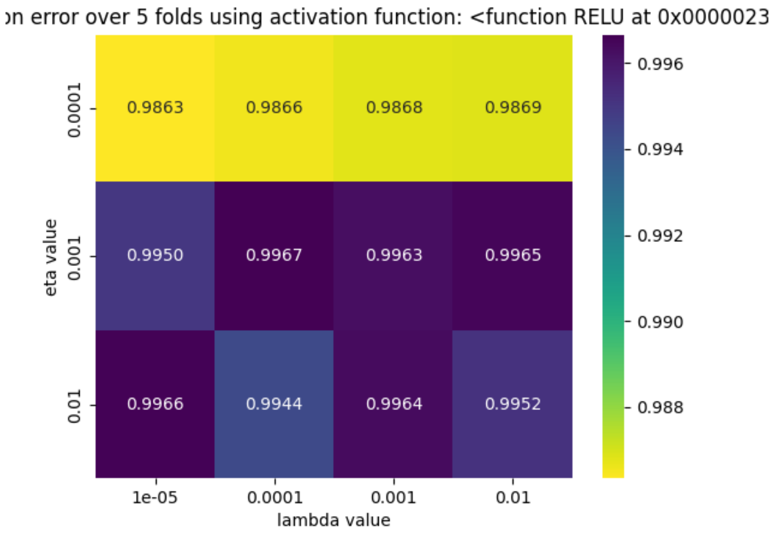
Figure 8: Heatmap of the best accuracy for given hyperparameters. In this case the full MNIST dataset is used.

Because we now have a multiclass-classification problem on our hands, some modifications are in order. Using the **softmax output function is essential**, as the output values are converted into probabilities. This property makes it generalize well to problems involving **multiple classes**. It also synergizes well with our choice for cost function, this being cross-entropy.

3.1.3 Finding the Optimal Activation Function

Before we get on with the architecture, we need to find the optimal function for the hidden layer for our case. Using the optimal parameters from before for hidden functions **RELU, LRELU and sigmoid** in the script "FFNN_MNIST_hidden.py", we get the following heatmaps :





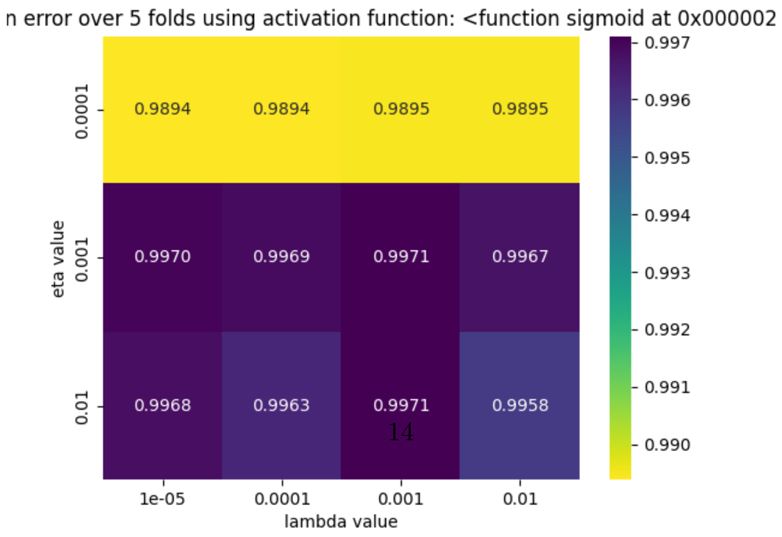
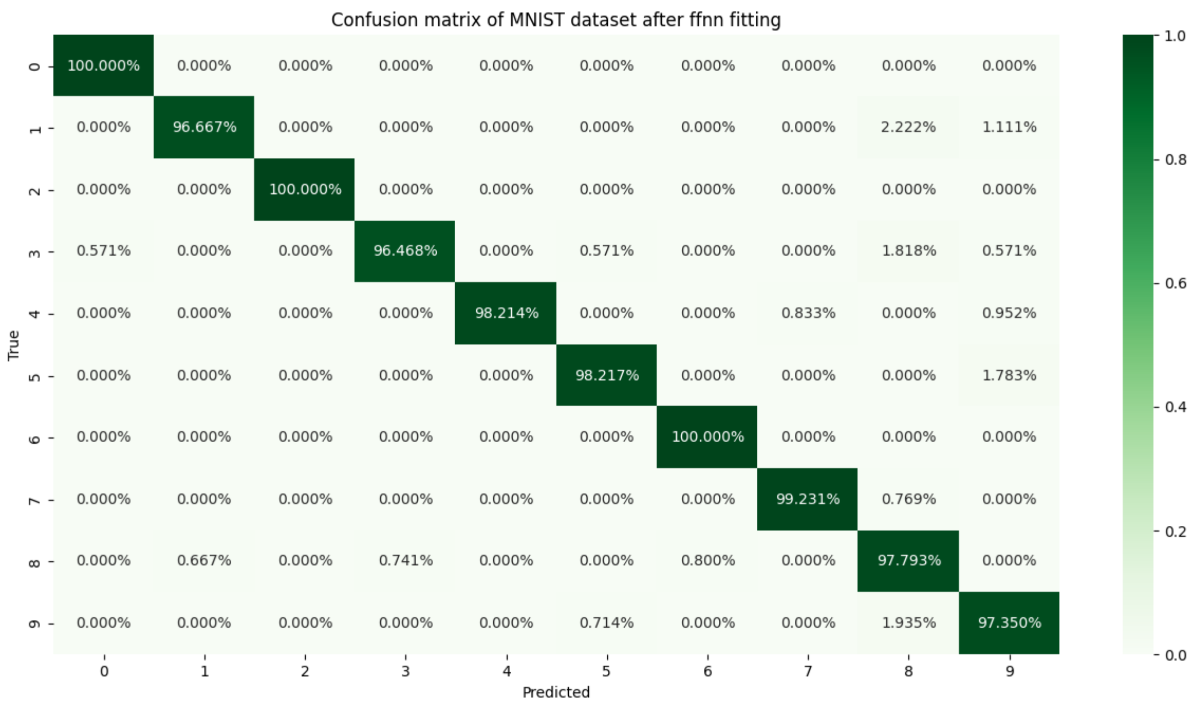


Figure 9: The three different activation functions for the hidden layer applied with the optimal hyperparameters from before

According to Figure 9, we conclude that the **sigmoid function is the best suitable activation function** for this dataset. Hence we will settle on this activation function as one of the optimized parameters.

3.1.4 Finding the Optimal Hidden Layer

Moving on to the architecture of the neural network: In the scripts "FFNN_MNIST_adam_nodes.py" and "FFNN_MNIST_adam_final.py" we find that the **optimal hidden layer is [92, 64, 64, 64]**. Implementing functionality for multi-class classification confusion matrices and running the script "FFNN_MNIST_adam_conf.py" results in the below confusion matrix :



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	100.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
1	0.000%	96.667%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	2.222%	1.111%
2	0.000%	0.000%	100.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%
3	0.571%	0.000%	0.000%	96.468%	0.000%	0.571%	0.000%	0.000%	1.818%	0.571%
4	0.000%	0.000%	0.000%	0.000%	98.214%	0.000%	0.000%	0.833%	0.000%	0.952%
5	0.000%	0.000%	0.000%	0.000%	0.000%	98.217%	0.000%	0.000%	0.000%	1.783%
6	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	100.000%	0.000%	0.000%	0.000%
7	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	0.000%	99.231%	0.769%	0.000%
8	0.000%	0.667%	0.000%	0.741%	0.000%	0.000%	0.800%	0.000%	97.793%	0.000%
9	0.000%	0.000%	0.000%	0.000%	0.000%	0.714%	0.000%	0.000%	1.935%	97.350%

Figure 10: The confusion matrix of the parameters and architecture found for our FFNN on the MNIST dataset.

The confusion matrix shows the results of the multi-class classification case, with a **total accuracy of 98,394%**

3.2 Gradient Boosting

3.2.1 Unscaled and Scaled dataset

At first we discuss whether or not scaling the data has a great impact in our results or not. In the previous projects, although **scaling can have a great impact** in balancing the effects of certain features, its impact on the data we used has **often been negligible**.

Here, we fix some parameters: the **number of estimators = 100**, the **learning rate = 0.3** and the **maximum depth = 3**. We will also settle on **Scikit-Learn's** implementation of Gradient Boosting for this section.

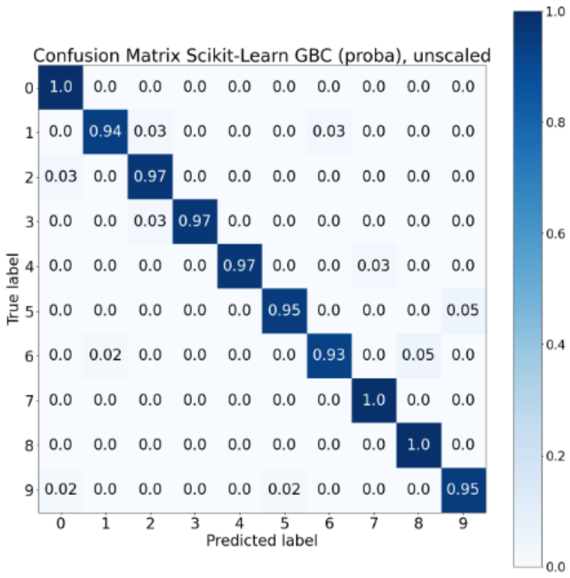


Figure 11: Confusion Matrix of our unscaled Digit Dataset. The overall accuracy of our model is about 97%

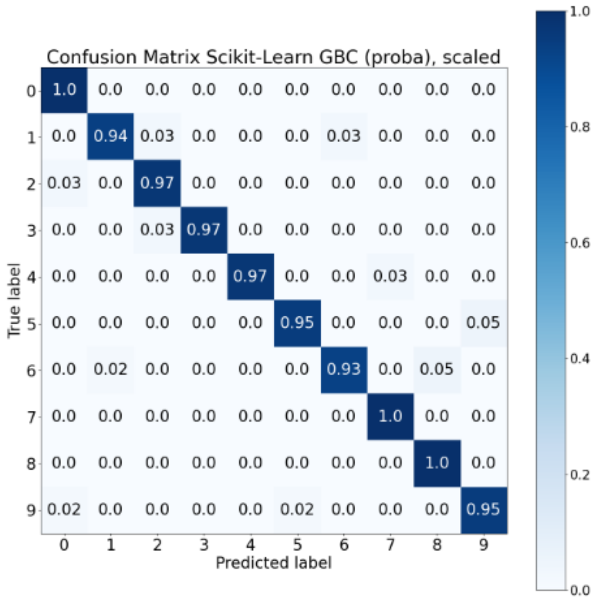


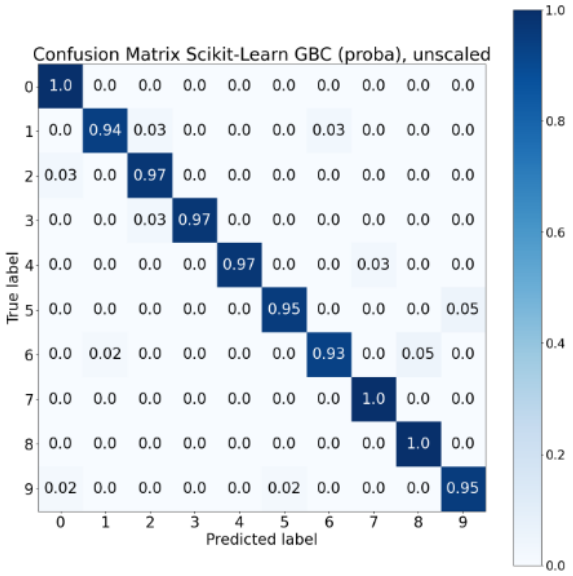
Figure 12: Confusion Matrix of our scaled Digit Dataset. The overall accuracy of our model is about 97%

The predictions of the GradientBoosterClassifier trained on scaled and unscaled digits data gives an overall good accuracy of 97% on the test data. Because the features in the design matrix are on a similar scale, the results of the model on scaled and unscaled data is **equal**.

3.2.2 Handmade and Scikit-Learn's versions

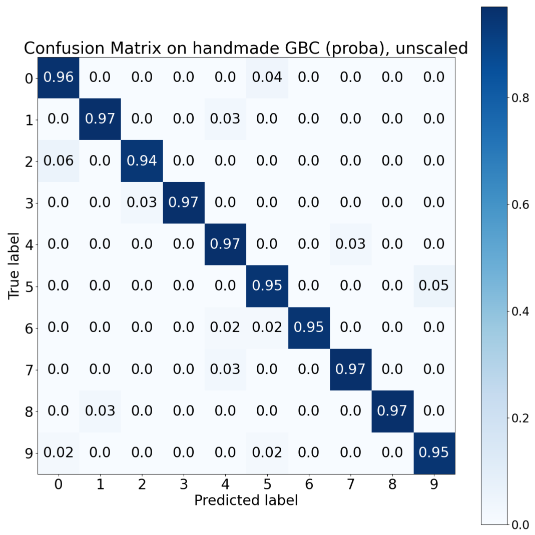
Scikit-Learn already has Gradient Boosting implemented, but we wished to get a better understanding of its implementation. The code from the handmade Gradient Boosting Classifier is is from scratch apart from the **Decision Trees**. In this section, we will try and discuss which method is the best suited for this problem.

Here, the fixed parameters are the same. We will also for now work on **unscaled data**, as we have concluded previously that scaling the data does not impact the results significantly.
The two models thus give the following results :



	0	1	2	3	4	5	6	7	8	9
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.94	0.03	0.0	0.0	0.0	0.03	0.0	0.0	0.0
2	0.03	0.0	0.97	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.03	0.97	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.97	0.0	0.0	0.03	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.95	0.0	0.0	0.0	0.05
6	0.0	0.02	0.0	0.0	0.0	0.0	0.93	0.0	0.05	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
9	0.02	0.0	0.0	0.0	0.0	0.02	0.0	0.0	0.0	0.95

Figure 13: Confusion Matrix of Scikit-Learn's model. The overall accuracy of our model is about 97%



	0	1	2	3	4	5	6	7	8	9
0	0.96	0.0	0.0	0.0	0.0	0.04	0.0	0.0	0.0	0.0
1	0.0	0.97	0.0	0.0	0.03	0.0	0.0	0.0	0.0	0.0
2	0.06	0.0	0.94	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.03	0.97	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.97	0.0	0.0	0.03	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.95	0.0	0.0	0.0	0.05
6	0.0	0.0	0.0	0.0	0.02	0.02	0.95	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.03	0.0	0.0	0.97	0.0	0.0
8	0.0	0.03	0.0	0.0	0.0	0.0	0.0	0.0	0.97	0.0
9	0.02	0.0	0.0	0.0	0.0	0.02	0.0	0.0	0.0	0.95

Figure 14: Confusion Matrix of our model made from stratch. The overall accuracy of our model is about 96%

These results are just slightly lower than Sklearn's Gradient Boosting results, on average an accuracy of 96%. This is possibly due to the use of different decision trees or optimization methods from Sklearn, but we will nonetheless consider Scikit-Learn's model as more accurate and settle on this model for the remaining of this project.

3.2.3 Finding the Optimal Learning Rate

In order to find the best learning rate for our model, we can simply fix the parameters the same way, but this time **iterate on the values of our learning rate** η. Just as before, we are sticking to **Scikit-Learn's model** on **unscaled data**.
A simple plot can give us this value :

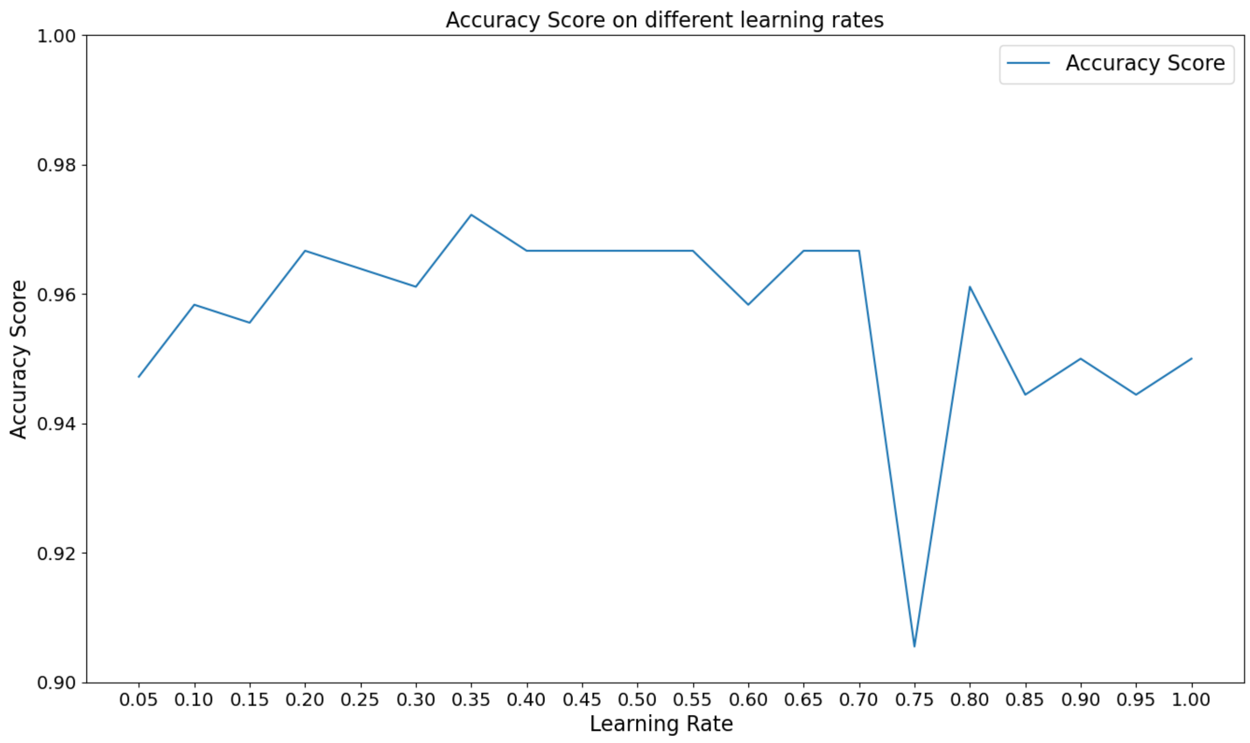


Figure 15: Plot giving the accuracy score of Scikit-Learn's model given the learning rate η.

Lowering the learning rate to 0.1 gives on average a slightly lower accuracy score of 96%. Lowering the learning rate further to 0.01, and the accuracy worsens. Also, using a higher learning rate than 0.3, for example 0.5, and the average accuracy stays the same. We can however see that the peak accuracy score is achieved with $\eta = 0.35$. With this learning rate value, we obtained the following confusion matrix :

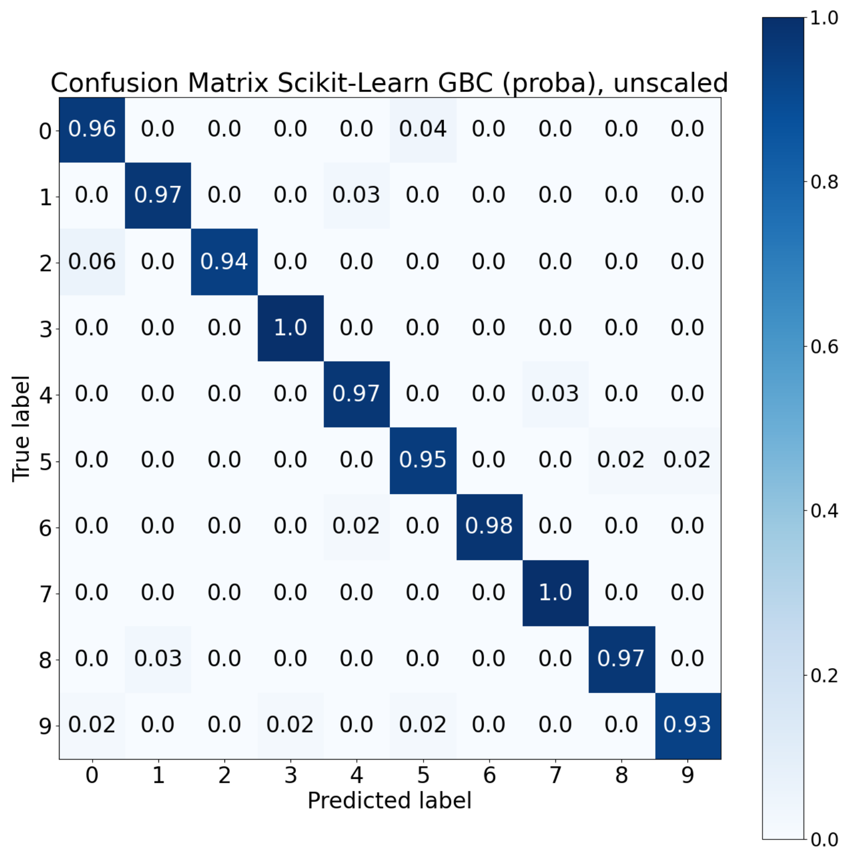


Figure 16: Confusion Matrix of Scikit-Learn's model, with the learning rate giving the best accuracy score.

The overall accuracy still remains at 97%, so the improvement is barely visible but is still a great asset nonetheless.

3.2.4 Finding the Optimal Maximum Depth

Just like in the previous subsection, we can find the best maximum depth for this problem by iterating on multiple values of them. Here we will study the maximum depth in the range [1,5], and discuss which depth is the best one here.
Once again, we can deduce this by a simple plot :

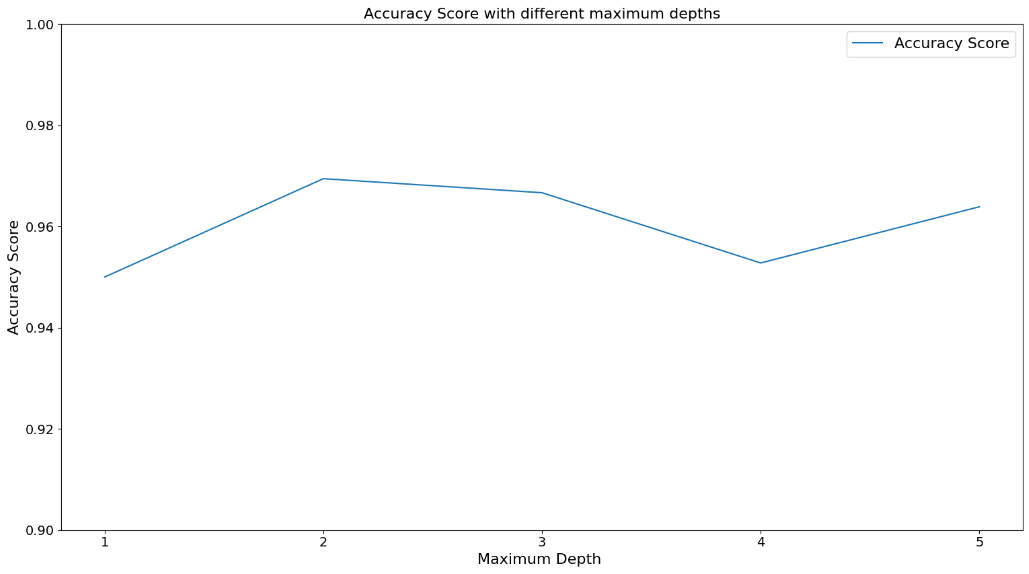
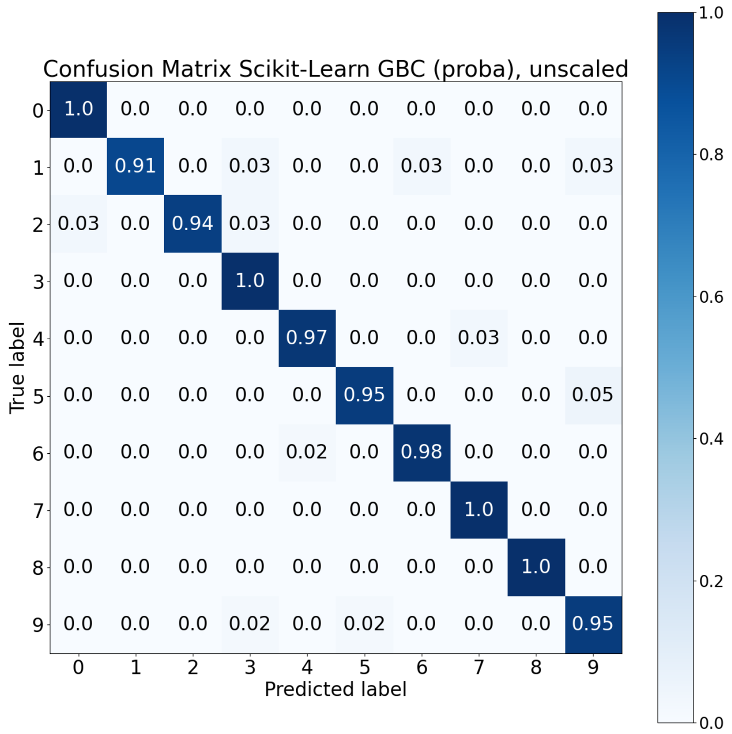


Figure 17: Plot giving the accuracy score of Scikit-Learn's model given the maximum depth of our model.

Figure 17 shows that the best accuracy score is achieved with a **maximum depth = 2**. Thus, by changing our depth to this optimal one, we obtain the following confusion matrix :



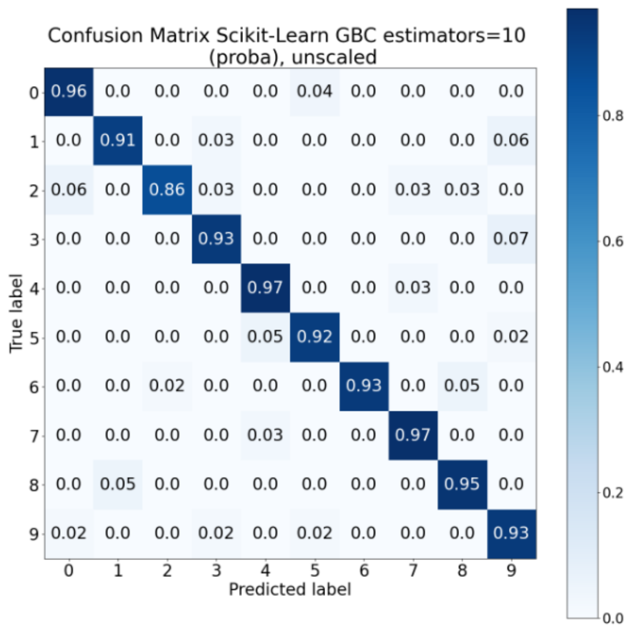
True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.91	0.0	0.03	0.0	0.0	0.03	0.0	0.0	0.03
2	0.03	0.0	0.94	0.03	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.97	0.0	0.0	0.03	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.95	0.0	0.0	0.0	0.05
6	0.0	0.0	0.0	0.0	0.02	0.0	0.98	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
9	0.0	0.0	0.0	0.02	0.0	0.02	0.0	0.0	0.0	0.95

Figure 18: Confusion Matrix of Scikit-Learn's model, with the maximum depth giving the best accuracy score.

The accuracy is again about 97%, so the change is not significant.

3.2.5 Finding the Optimal Number of Boosting Steps

This approach is about the same idea as for the learning rate, since we can just fix parameters and change the number of boosting steps. This is done by modifying the **number of estimators**.
Here, we will simply study the case of **10 and 1000 steps**, as iterating as we did previously may end up in a too long computation.



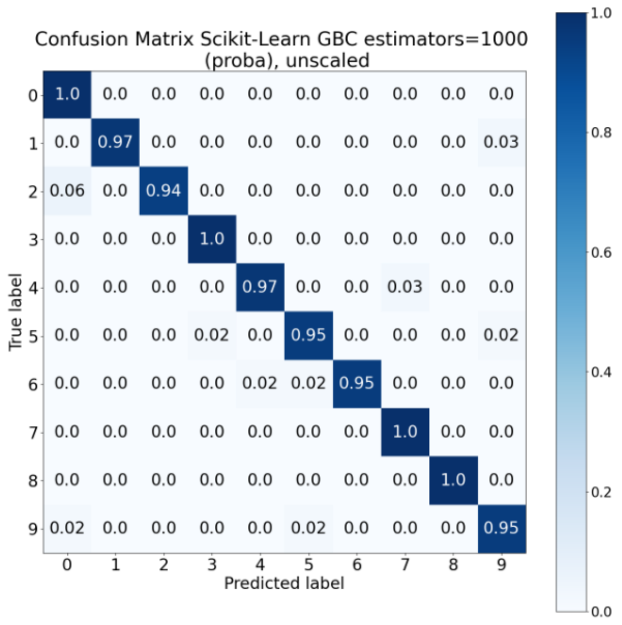


Figure 19: Confusion Matrix of Scikit-Learn's model when using 10 iterations. The overall accuracy of our model is about 93%

Figure 20: Confusion Matrix of Scikit-Learn's model when using 1000 iterations. The overall accuracy of our model is about 97%

We find, as expected, that lowering them will **give worse results** in Figure 19 and increasing them gives **about the same results** while taking longer to run in Figure 20. Hence, we will **stick to 100 iterations** as it gives a good **trade-off between accuracy and time computation.**

3.2.6 Analyzing the ROC Curve

If we now look at the ROC Curve with our best performing parameters (iterations = 100, $\eta = 0.35$, max depth = 3, Scikit-Learn's model, unscaled data), we obtain the following graph :

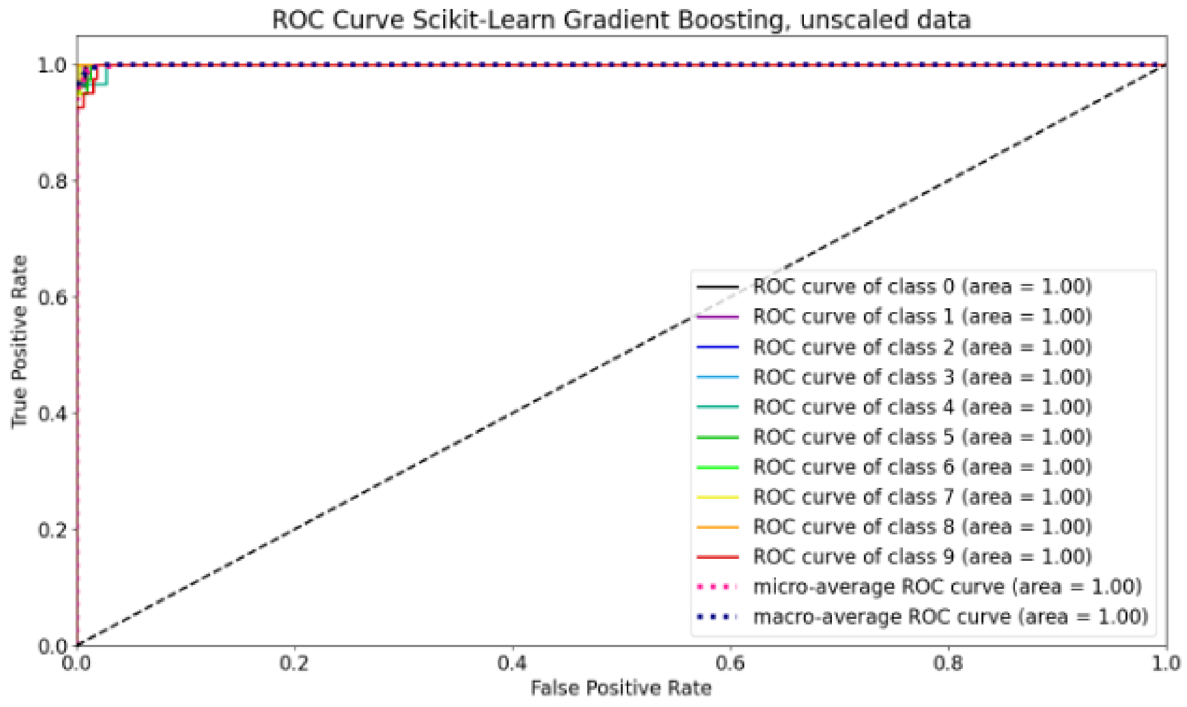


Figure 21: ROC Curve of Scikit-Learn's model with our best performing parameters.

We can clearly see that the ROC Curve **converges to 1 extremely quickly** for every single class, indicating a **very well fit** of the data.

The ROC Curves of our other studied parameters have also been plotted, but as they are very similar to this one graph, they are not shown here, but are available in our *GitHub repository*.

3.3 Logistic Regression

We have already implemented our handmade version of Logistic Regression in our **previous report**. Since we have also stated in that same report that **Scikit-Learn's implementation performs better than ours**, we will strictly **focus on the latter** in this project.

Scikit-Learn's version also finds itself the best parameters for the problem, hence we simply need to run in and get its results. Afterward we will see how it holds up to the other twos.

3.3.1 Unscaled and Scaled

Scikit-Learn's optimization is independent to whether or not the data is scaled, hence they can produce different results. Thus, we compare the confusion matrix of Logistic Regression on both unscaled and scaled data, as we did previously with Gradient Boosting.3.2.1 :

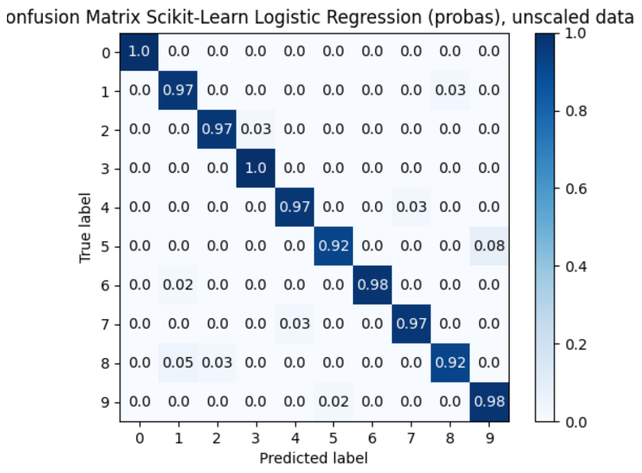


Figure 22: Confusion Matrix of our unscaled Digit Dataset. The overall accuracy of our model is about 97%

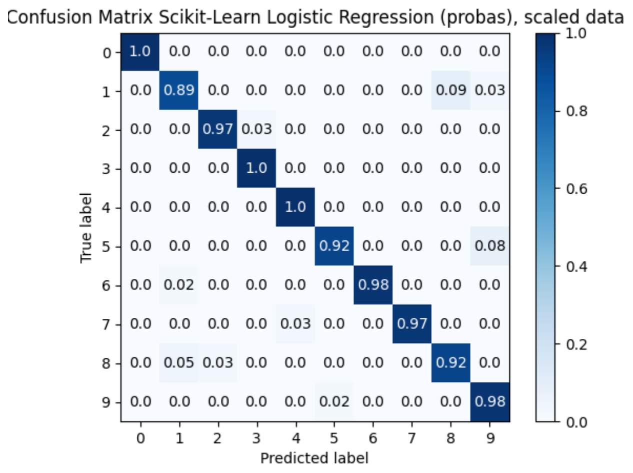


Figure 23: Confusion Matrix of our scaled Digit Dataset. The overall accuracy of our model is about 96%

Once again, we realize that **scaling the data does not matter for this dataset**. Here, it even **worsen** the results as the accuracy score drops by 1% by doing so. Hence the rest of this study will be done on **unscaled data**.

Nonetheless, the accuracy score is still a pretty good result, even matching the ones done by **Gradient Boosting**.

3.3.2 Analyzing the ROC Curve

The ROC Curve of Logistic Regression gives the following plot

Just like before, the values quickly jumps to 1, indicating of a great fit of the data. Since Logistic Regression has both similar accuracy score and ROC Curve, we can expect them to be **very similar for this precise dataset**.

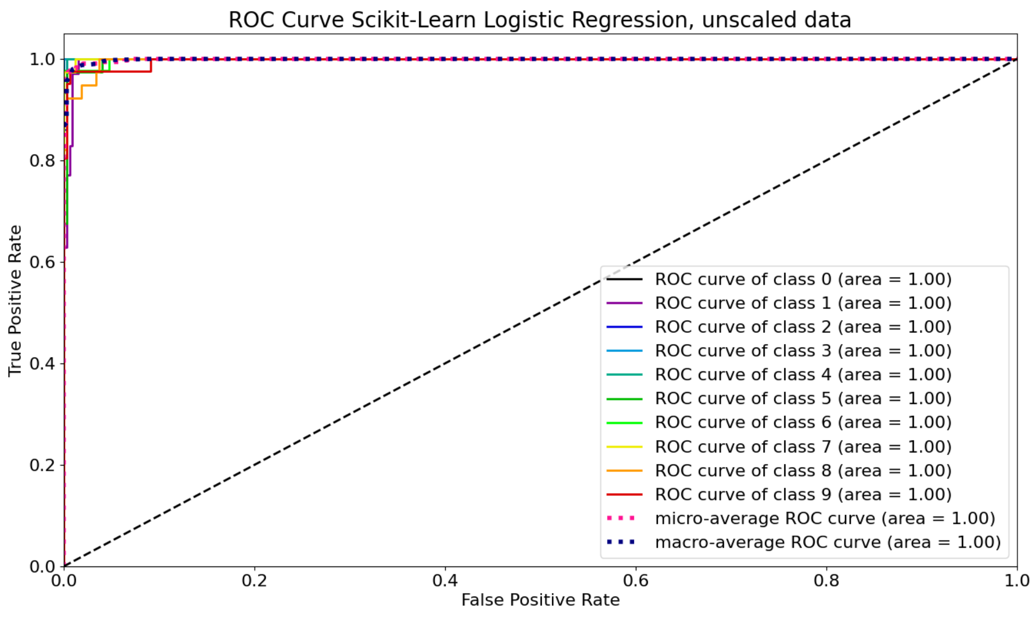


Figure 24: ROC Curve of Scikit-Learn's model with Scikit-Learn's implementation of Logistic Regression.

4 Comparison with Existing Papers

In this section, we will present other reports or studies on that same dataset, and will compare our results with them. More importantly, we will point out the eventual similarities and differences and, if possible, explain why.

The results are mostly taken from university reports and blogs, and will all be referenced in the following subsections.

4.1 Feed Forward Neural Network

We will start by comparing our results with our FFFNN with other studies. Our first paper analyses the Digit Dataset with **Artificial Neural Networks** [4], FFNN included.

In this paper, the authors try to find the most optimal parameters in order to obtain the best possible prediction score, and end up with a accuracy score of **98.96%**, achieved with **iterations = 400, η = 0.01, 8 hidden layers and 90 nodes per hidden layers**.

A second study on the matter can be found on **GeeksForGeeks** [8], this time achieving an **accuracy score of 91.47%**, way inferior to our results.
This can however be explained by the fact that they chose to use **only the first 1000 instances as training set**, leaving the remaining 4620 instances as test set. The training set thus makes up for 18% of the overall set, so there is a great chance of **underfitting** in these circumstances.
They also decided to set some arbitrary parameters without prior theorization nor optimization, making it unlikely

that these parameters are the optimal ones.

4.2 Gradient Boosting Classifier

Now, we will focus on the Gradient Boosting Classifier, and how our version compares to those applied by other researchers.

A blog study has been conducted by **Sunny Solanki on CoderzColumn** [16], introducing numerous boosting methods on Scikit-Learn's Digit Dataset, Gradient Boosting included.
Indeed, when applying the **GradientBoostingClassifier** method of Scikit-Learn's, he obtains an accuracy score of **95.6%**, a shy difference from the 97% accuracy concluded with our method.
Once again, this is due to the fact that there has been no prior optimization of the parameters used for the model. Indeed, the paper fixes the **learning rate** $\eta = \mathbf{0.1}$, and we have discussed that fixing this value to 0.35 was the most optimal, hence the difference.

There is also a small code repository on GitHub using Gradient Boosting on this particular dataset. When calculating the accuracy score, he obtains a score of 93%, which is again inferior to our results. This difference here is because the number of steps has been fixed to **n_estimators = 10**, which leads to underfitting as it is way too small to converge to a good value. However, we obtained the same accuracy when studying the same parameters.

4.3 Logistic Regression

Lastly, a small study has been done on this dataset using Logistic Regression on **Medium** by Michael Galarnyk [7]. Here, they achieve an accuracy score of **95.3%**, which is quite close from our results. This is expected since they also let their model optimize the parameters itself. The only difference relies in the test dataset's size, which is **slightly bigger in Michael's analysis**, so there could be a slight underfit on his part.

4.4 Scikit-Learn's Baseline

Scikit-Learn also gives a baseline accuracy score for multiple methods along with the dataset's description : [1]
As we can see in this graph, a Neural Network should be around 97% accuracy, while Logistic Regression is around 96%. Those are two metrics that we have **surpassed during this project**, since we did our best to optimize our parameters, whereas this baseline only suggests a minimum result for our models. There is however no baseline for Gradient Boosting, but we can assume that it should not be far from **XGBoost's results**, which are about 96% as well, a value that has also been exceeded in our analysis.

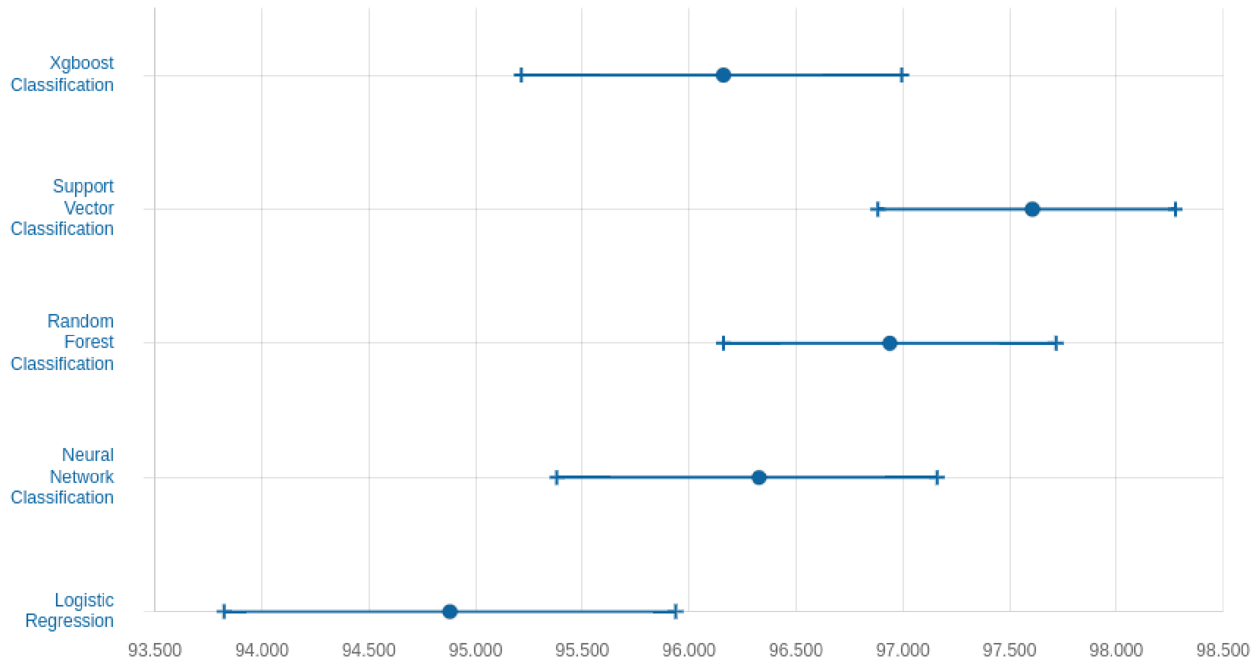


Figure 25: Scikit-Learn's baseline accuracy for multiple models

5 Conclusion

During this project, we have observed that on a multiclass classification problem, although it is deemed more difficult to analyze, the prediction methods explored during the semester are **able to adapt and give good results**. The accuracy scores prove that a 10-class problem is **not as big enough of an obstacle** to severely reduce the ability to predict the correct output. Out of the observed methods, **the FFNN is again the one performing the best and by a far margin**. Although this dominance was not very visible in a binary classification problem, this deeper problem shows how well this method adapts compared to other methods, and proves once more that a **Neural Network is the best suitable model for deep learning problems**. It is very likely that they will remain a stable models for nowadays problems, and even more complex ones in the future.

6 Appendix

Once again, all of our code can be found in our GitHub Repository, and you can also find our previous report here, as it has been cited multiple times during this report.

References

[1] E. Alpaydin and C. Kaynak. *Optical Recognition of Handwritten Digits.* Accessed 17 Dec 2023. June 1998. URL: https://archive.ics.uci.edu/dataset/80/optical+recognition+of+handwritten+digits.

[2] Matt Bowers. *Gradient Boosting Multi-Class Classification from Scratch.* Accessed 16 Dec 2023. URL: https://python-bloggers.com/2023/10/gradient-boosting-multi-class-classification-from-scratch/#google_vignette.

[3] Master's in Data Science. *What Is a Decision Tree?* Accessed 16 Dec 2023. URL: https://www.mastersindatascience.org/learning/machine-learning-algorithms/decision-tree/#:~:text=A%20decision%20tree%20is%20a,that%20contains%20the%20desired%20categorization..

[4] Toufik Datsi, Khalid Aznag, and Ahmed El Oirrak. "Digit recognition using decimal coding and artificial neural network". In: (Jan. 2022). Accessed 17 Dec 2023.

[5] Google Developers. *Classification: ROC Curve and AUC.* Accessed 16 Dec 2023. URL: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

[6] Google Developers. *Machine Learning Glossary.* Accessed 16 Dec 2023. URL: https://developers.google.com/machine-learning/glossary.

[7] Michael Galarnyk. *Logistic Regression using Python (scikit-learn).* Accessed 17 Dec 2023. Sept. 2017. URL: https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a.

[8] GeeksForGeeks. *Recognizing HandWritten Digits in Scikit Learn.* Accessed 17 Dec 2023. Apr. 2023. URL: https://www.geeksforgeeks.org/recognizing-handwritten-digits-in-scikit-learn/.

[9] Morten Hjorth-Jensen. *Gradient Boosting Explained – The Coolest Kid on The Machine Learning Block.* Accessed 16 Dec 2023. URL: https://www.displayr.com/gradient-boosting-the-coolest-kid-on-the-machine-learning-block/.

[10] Morten Hjorth-Jensen. *Pruning the Tree.* Accessed 16 Dec 2023. URL: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week46.html#pruning-the-tree.

[11] Morten Hjorth-Jensen. *ROC curve.* Accessed 16 Dec 2023. URL: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week45.html#roc-curve.

[12] StatQuest with Josh Starmer. *Gradient Boost Part 4 (of 4): Classification Details.* Accessed 16 Dec 2023. URL: https://www.youtube.com/watch?v=StWY5QWMXCw.

[13] Monis Khan. *What are pseudo residuals?* Accessed 16 Dec 2023. URL: https://www.csias.in/what-are-pseudo-residuals/#:~:text=Pseudo%20residuals%20are%20intermediate%20error,actual%20value%20of%20target%20variable..

[14] Knock Knock. *Knock Knock Funny Flowcharts to Help You Make the Right (Irreverent) Decisions.* Accessed 16 Dec 2023. URL: https://www.pinterest.com/pin/71002131601111582/.

[15] Aratrika Pal. *Gradient Boosting Trees for Classification: A Beginner's Guide.* Accessed 16 Dec 2023. URL: https://medium.com/swlh/gradient-boosting-trees-for-classification-a-beginners-guide-596b594a14ea.

[16] Sunny Solanki. *Scikit-Learn - Ensemble Learning: Boosting.* Accessed 17 Dec 2023. May 2020. URL: https://coderzcolumn.com/tutorials/machine-learning/scikit-learn-sklearn-ensemble-learning-boosting#2.