FYS-STK3155 Project 3

Lasse Pladsen, Parham Qanbari, & Sander V. Vattøy

December 1, 2023

Abstract

I. INTRODUCTION

Two possibilities exist: either we are alone in the Universe or we are not. Both are equally terrifying.

Arthur C. Clarke

In this project we will use machine learning methods to look for exoplanets. The best place to look for extraterrestrial life is on other planets. There are thousands of planets orbiting other stars in the universe, called exoplanets. But, given the large distance in our universe it makes it difficult to simply "look" for planets. There are many features and methods in astronomy that could be used to detect these exoplanets. One of which is using the flux dip of the star.

The data set we will be using contains the flux of 5087 stars in which only 37 stars have exoplanets orbiting them (Winterdelta, 2017). The flux data can be used to detect whether a exoplanet is orbiting the star, this is called the exoplanet transit method (Wilson, 2017). We will use this data in different machine learning methods and see what methods is best suited for detecting exoplanets. The methods we will we use with the dataset is Logistic regression, Feed Forward Neural Network (FFNN), decision tree, and extreme gradient boosted decision tree.

II. THEORY

A. Dataset

The data is from NASA's Kepler space telescope. The cameras of the satellite stare at one area of space for 80 days (Winterdelta, 2019). Thus for 80 days light of one area of space is collected. We can use this continuous data sequence and look for dips in the light flux from stars. This is called the exoplanet transit method (Wilson, 2017). Essentially, the dip in flux from a star could mean that there is a planet orbiting it. However, it could also mean its another object passing, maybe a very large asteroid, or maybe even disturbances in the camera. Nevertheless, we will use this method to train our different machine learning models and eventually see how well the methods are able to learn whether a flux dip implies an exoplanet in orbit.

The data set is formatted as a table/matrix where each row is a data sample (a star) and each column/feature is a flux measurement for that row's star. The data contains 3198 such features for each star. The total data set has a total of 5657 stars where 42 of them have a confirmed exoplanet in orbit. The data set comes pre-split into a training set with 5087 stars where 37 of them have a confirmed exoplanet, and a testing set with 570 where 5 of them have a confirmed exoplanet.

B. Scaling

We apply a standardization scaling to our data set. This is a scaling method that, for each feature, subtracts the whole feature by its mean, then divides it by the standard deviation:

$$x_{ij} = \frac{x_{ij} - \bar{x}_{:j}}{\sigma(x_{:j})}$$

where $\bar{x_{:j}}$ and $\sigma(x_{:j})$ are respectively the mean and the standard deviation of the feature column j defined as

$$\bar{x}_{:j} = \sum_{i}^{n_{rows}} x_{ij}$$

$$\sigma(x_{:j}) = \frac{1}{n_{rows}} \sqrt{\sum_{i}^{n_{rows}} (x_{ij} - \bar{x}_{:j})^2}$$

This method ensures that each feature has a mean value of zero and a standard deviation of one.

C. Measuring prediction performance

C1. Confusion matrix

The confusion matrix is a matrix to show the different sub-accuracies like the false negative, true negative, false positive, and true positive in a single matrix:

$$\text{Confusion matrix} = \begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix} \tag{1}$$

where each of the metrics are defined as below:

$$TN = \text{true negative} = \frac{\text{No. correct negative predictions}}{\text{Total No. negative predictions}}$$

$$FN = \text{false negative} = \frac{\text{No. false negative predictions}}{\text{Total No. negative predictions}}$$

$$TP = \text{true positive} = \frac{\text{No. correct positive predictions}}{\text{Total No. positive predictions}}$$

$$FP = \text{false positive} = \frac{\text{No. false positive predictions}}{\text{Total No. positive predictions}}$$

In this sense a negative prediction is one that predicts the class 0, and positive is the prediction of 1.

C2. Accuracy

For classification problems the accuracy is defined as:

$$\begin{aligned} Accuracy &= \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \\ &= \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n} \end{aligned} \tag{2}$$

where I is the indicator function that simply checks if they are equal:

$$I(t = y) = \begin{cases} 1, & t = y \\ 0, & t \neq y \end{cases} \tag{3}$$

this can also be written out as

$$Accuracy = \frac{TN + TP}{FN + TN + FP + TP} \tag{4}$$

D. Logistics regression

Logistic regression is a suitable method to use for this particular problem of finding whether there is a planet in orbit or not. For each data point get squeezed between 0 and 1 through the Sigmoid function which is defined as (Hjorth-Jensen, 2023)

$$f(x) = \frac{1}{1 + e^x}, \tag{5}$$

The resulting values ranges from 0 to 1,

$$f(x) \to [0, 1] \tag{6}$$

E. Feed Forward Neural Network

For the neural network method we will test Feed Forward Neural Network (FFNN). A FFNN has an input layer which the data is fed into as a matrix $\mathbf{X}$, at least one hidden layer, and at last a output layer (Pladsen, Qanbari, & Vattøy, 2023). The basic schematic is illustrated in figure (1).

In the hidden hidden layer the input data $\mathbf{X}$ is multiplied with weights $\mathbf{W}_l$ and a bias $\mathbf{b}_l$ is added resulting in the equation (7) (Pladsen et al., 2023).

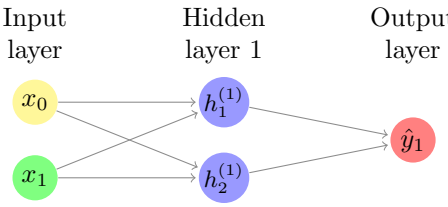

Figure 1: Neural network schematic.

$$\mathbf{z}_l = \mathbf{X}\mathbf{W}_l + \mathbf{b}_l \tag{7}$$

The output of the function $\mathbf{z}_l$ in the hidden layer is passed through an activation function resulting in a function (8)

$$\mathbf{a}_l = f(\mathbf{z_l}) \tag{8}$$

The activation function $\mathbf{a}_l$ is then multiplied by weights in the output layer $\mathbf{W}_L$ and a bias is added $\mathbf{b}_L$ resulting in the function $\mathbf{z}_L$ which could also be passed through an activation function depending on the problem at hand (Pladsen et al., 2023).

E1. Activation functions

Activation functions are an essential component of a neural network. Inspired from neurons in biological systems where neuronal signalling is dependent on the activation level. A neuron does not fire upon any activation, it required a certain threshold before firing (Hjorth-Jensen, 2023; Pladsen et al., 2023). Activation functions in neural networks are set perform in a similar way. This adds important features to the model, such as improving the training convergence of the network (Dubey, Singh, & Chaudhuri, 2021). Without activation functions, a neural network would be simply be a regression model (GeeksforGeeks, 2023). In other words, the activation functions do non-linear transformations on the input data that allows the network to learn the abstract features of the data (Dubey et al., 2021; GeeksforGeeks, 2023; Pladsen et al., 2023).

There are a myriad of activation functions to chose from. However, the most reliable and suitable functions for the purpose for the classification problem in this project is the rectified linear unit (ReLU) functions.

The ReLU (Rectified linear unit) function (figure 2) is defined as (Pladsen et al., 2023):

$$f(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{9}$$

Its values range from 0 to infinity,

$$a_l = f(\mathbf{z}_l) \to [0, \infty] \tag{10}$$

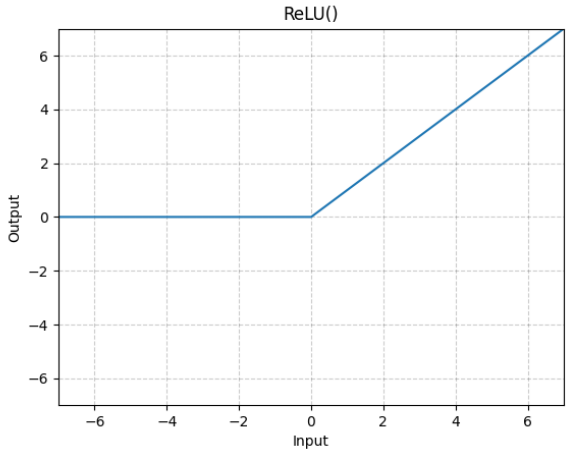

Figure 2: The ReLU function, from pytorch.org.

The ReLU function has an limitation which is the dying of neurons caused by negative values. If the amount of negative values are significant in the data set then, most of the data is set to zero in the activation function (Dorsaf, 2021; Dubey et al., 2021). However, as explored in (Pladsen et al., 2023) it proved to be computationally efficient and lead to high prediction accuracy.

Another function is the Sigmoid function (figure 3) as defined in equation (5). However, as explored in (Pladsen et al., 2023) the Sigmoid function has some limitations. One of which is the saturation of neurons when the activation becomes 0 and 1. The derivative of the Sigmoid function is:

$$f'(x) = f(x)(1 - f(x))$$

thus for activations that are 0 or 1 the derivative becomes zero and training of the neuron essentially stops (Dorsaf, 2021). Thus the Sigmoid function could hinder training and the calculation of optimal parameters (Pladsen et al., 2023; Dorsaf, 2021).

The last activation function we test is the hyperbolic tangent function (tanh) plotted in figure 4, which is defined as:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{11}$$

Its values range from -1 to 1,

$$f(x) \to [-1, 1] \tag{12}$$

Tanh also has the same limitation as Sigmoid, which is that it also cause saturation of neurons for values -1 and 1.

E2. Backpropagation

We use the algorithm known as backpropagation to train our feed-forward neural network. For a network we want

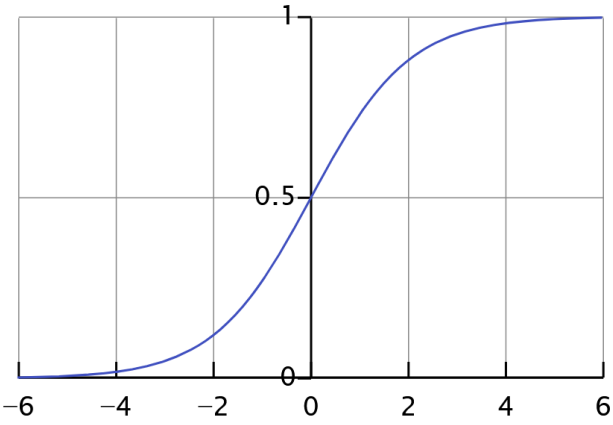

Figure 3: The Sigmoid logistic function, from wikipedia.

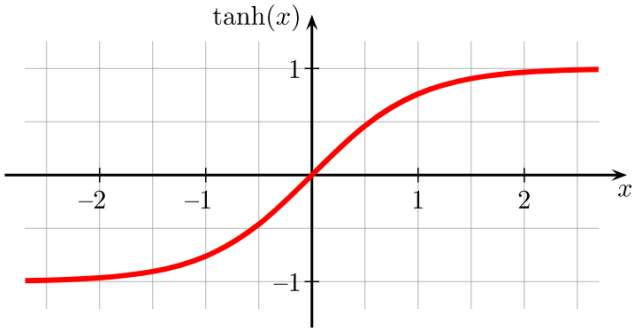

Figure 4: Plot of the hyperbolic tangent function, from wikipedia.

to optimize the weights $\mathbf{W}$ and the biases $\mathbf{b}$ in the different layers by using the error for the output layer L generally defined as **??** and for a general layer l defined as **??**. More about this can be found in (Pladsen et al., 2023), but `Scikit-learn` can also be used to easier analyse datasets.

$$\begin{aligned}\boldsymbol{\delta}^L &= \frac{\partial C}{\partial \mathbf{z}} = \frac{\partial \mathbf{a}}{\partial \mathbf{z}}\frac{\partial \mathbf{C}}{\partial \mathbf{a}} \\ &= f'(\mathbf{z^L})\frac{\partial \mathbf{C}}{\partial \mathbf{a}}\end{aligned}$$

$$\begin{aligned}\boldsymbol{\delta}^l &= \boldsymbol{\delta}^{l+1}(\mathbf{W^{l+1}})^T\frac{\partial \mathbf{a^l}}{\partial \mathbf{z^l}} \\ &= \boldsymbol{\delta}^{l+1}(\mathbf{W^{l+1}})^T f'(\mathbf{z^l})\end{aligned}$$

F. Decision Classification tree

A classification tree is a machine learning method that splits the data based on certain conditions. The basic schematic of a binary tree is illustrated in figure (5).

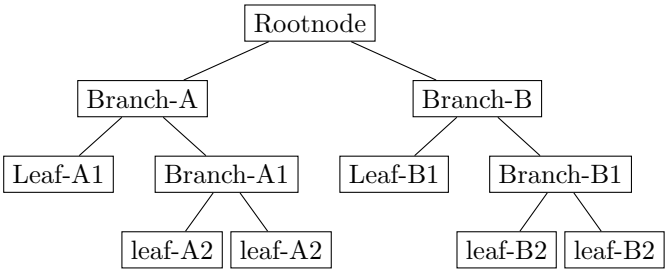

Figure 5: Binary classification tree schematic

As shown in figure (5) the tree starts with a root node and splits into two branches, also called internal nodes (Hjorth-Jensen, 2023; IBM, n.d.). The internal node contains an condition used to take a decision on how to classify the features of the data. For instance, lets say we classifying exoplanets, and the root node asks "Is there a star?" with conditions yes/no. If "no" we move to Branch-A. Where the node asks "Is there planet in orbit", if "yes" then we move down to Leaf-A1. The leaf represents an instance of a classification, which in our case it could be "exoplanet", i.e the conditions that led to this classification in Leaf-A1 could be used to detect exoplanets based on the data.

This algorithm is employed from the top of the tree and until all of the data has been classified (IBM, n.d.). Decision trees are prone to overfitting, therefore it is common to set a threshold for how deep the classification goes (IBM, n.d.). Thus it might not be beneficial to make each split optimal for every classification. In the example above the splitting was decided upon simple yes/no, however, for a data set containing flux observation, the decision must be numeric.

G. Splitting Nodes

Two common node splitting methods are the Gini index and information entropy (Hjorth-Jensen, 2023).

The Gini index is defined as (13):

$$g = \frac{1}{N_m}\sum_{k=1}^{K} p_{mk}(1-p_{mk}) \tag{13}$$

Where N_m is the number of observations of class m, p_{mk} is the probability of the data being classified as class k (Hjorth-Jensen, 2023; T, 2021):

$$p_{mk} = \frac{1}{N_m}\sum_{x_i} I(y_i = k) \tag{14}$$

The gini index bases the splitting of nodes on the probability of a random data point being misclassified if it is randomly assigned to a node (Dash, 2023).

Another method is the information entropy defined as:

$$s = -\sum_{K=1}^{K} p_{mk} log p_{mk} \tag{15}$$

The information entropy bases a split on the information gain, which is a measure of the purity of the possible classification (Dash, 2023). This is based of thermodynamic entropy from physics, and it suits well for decicion trees because it can be used to split a tree based on whether the splitting causes a information gain, or in other words, a purer classification (Dash, 2023).

H. Finding optimal depth

Starting from the the root node and splitting the three downward can make a large tree with many branches. How large could the tree become? In theory the splitting could go on until all branches represent a classification (Hjorth-Jensen, 2023). However, for increasing tree complexity the classification becomes less pure, meaning fewer data points end up branches and have a lower purity (IBM, n.d.).

In order to avoid overfitting we could find the optimal depth with respect to the accuracy of the model. It is also possible to develop a complex tree and then prune the tree back a few depths. In our model we test various depths and use a grid search for the optimal depth and regularization term.

H1. plain algorithm

I. Gradient tree boosting

Gradient tree boosting is a method using gradient descent/steepest descent for decision trees. This gradient boosting uses a decision tree as a weak [???]

I1. Extreme gradient boosting

??? XGBoost ??

III. METHODS

Our star dataset (A) is already split into a training and testing set, however we need to preprocess it. The stars with a confirmed exoplanet are labeled with the label 2, while the stars without confirmed exoplanets have the label 1. We first change these labels to 0 and 1 so we can easily use the Sigmoid function to get the prediction outputs between $[0, 1]$. The label 1 means the star has a confirmed exoplanet.

The data set classes are also extremely unbalanced; the training set contains 5050 non-exoplanet stars, but only 37 confirmed exoplanet stars. This will heavily impact our training and predictions. To solve this we use a technique called Synthetic Minority Over-sampling Technique (SMOTE) (Chawla, Bowyer, Hall, & Kegelmeyer, 2002). For this we use the Python library Imbalanced-learn: `imblearn.over_sampling.SMOTE`. This balances our two classes 0 and 1. We choose to do this to just the training set and then re-split it into a new training and testing set. One could also include the original testing data by combining the data before preprocessing it. We use Scikit-learn's `sklearn.model_selection.train_test_split` with a testing size of 20%.

For the different classification methods we test out scaling the data before training, to be able to see if it improves performance. For this we choose to use standardization scaling from Scikit-learn's `sklearn.preprocessing.StandardScaler`.

We use the accuracy (2) to grade all our predictions using Scikit-learn's `sklearn.metrics.accuracy_score`. In addition, we plot the confusion matrices (**??** for the results using `sklearn.metrics.confusion_matrix`. These are applied to the prediction on the testing set after being trained on the training set.

A. Logistic regression

From Scikit-learn we use `sklearn.linear_model.LogisticRegression`, and apply a L2 regularization penalty term λ with 1000 epoch iterations. To find out if scaling the data improves the prediction we start off with $\lambda = 10^{-4}$.

We plot the testing set accuracy with respect to λ to find the optimal value. Here we choose to use 100 epochs to reduce the computation time. Using the found optimal value we then do our final prediction with 1000 epochs.

B. Classification neural network

From Scikit-learn we now use `sklearn.neural_network.MLPClassifier` for our classification network. We use the ADAM method (Pladsen et al., 2023) for scaling the learning rate.

The parameters we need to optimize are the L2 regularization λ, the initial learning rate η_0, the activation function, the number of batches $n_{batches}$, number of hidden layers n_{layers} and the number of nodes in them n_{nodes}. The last two can be written as a vector of the number of nodes for each layer

$$n_{nodes} = (n_{node1}, n_{node2}, ...)$$

Initially to test if scaling improves performance we use the parameters

$$\begin{aligned} \lambda &= 0.001 \\ \eta_0 &= 0.1 \\ n_{batches} &= \text{auto (Scikit-learn)} \\ \text{activation} &= \text{ReLU (9)} \\ n_{epochs} &= 100 \end{aligned}$$

First we try out different numbers of hidden layers in our network. Secondly, we test the different activation functions Sigmoid (5), Identity ($f(x) = x$), and finally tanh (11). Thirdly, we do a grid search plotting accuracy to η_0 and λ. Lastly, we do same for $n_{batches}$ and n_{nodes} for the optimal number of layers.

Using the optimal parameters we do our final prediction with 1000 epochs.

C. Classification tree

From Scikit-learn we use `sklearn.tree.DecisionTreeClassifier` for our classification tree.

Initially we use a max tree depth of 5 using the Gini index (13). Afterwards we test the information entropy (15) method, and [logloss??]

Using the optimal splitting method we plot the accuracy as function of the maximum tree depth, then use that found value to make our final prediction.

D. Extreme gradient boosted tree

For extreme gradient boosting we use the library XGBoost and use the `xgboost.XGBClassifier` classification tree.

Initially we use a maximum tree depth of 5 with 100 boosting rounds to test out scaling, with no regularization. Afterwards we create a grid plot to find the optimal values for these two parameters. We also want to find the optimal λ for L2 regularization if it improves the performance, we use these optimal parameters to make our final prediction.

IV. RESULTS

A. Logistic regression

When testing if scaling have any impact for the chosen parameters, we find that with and without scaling we respectively have accuracy scores 0.978 and 0.898.

We then plot the accuracy with respect to the L2 parameter shown in figure 6, and from this find the optimal parameter $\lambda = 10^{-5}$. Using this we find the accuracy result 0.981, and the confusion matrix in figure 7.

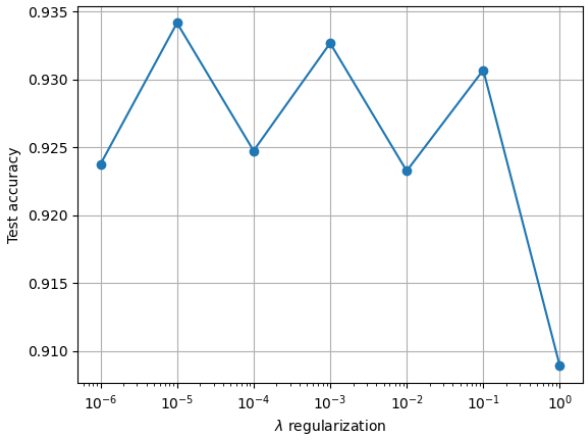

Figure 6: ...

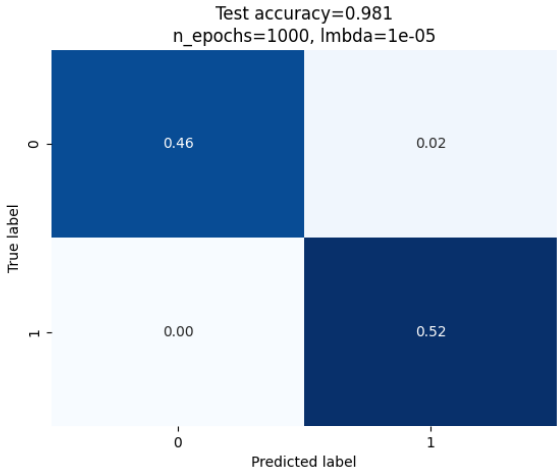

Figure 7: ...

B. Classification neural network

We again test with and without scaling and find respectively the accuracies 0.969 and 0.980 for the chosen starting parameters and activation function. We then found that for one hidden layer we only obtained an accuracy of 0.482. We then tried out the different activation functions and found the following accuracies

Identity : 0.746

Sigmoid : 0.808

Tanh : 0.520

Thereafter we moved on to find the neural networks optimal parameters. The optimal η and λ values are presented in figure 8 and the optimal number of batches and nodes in figure 9.

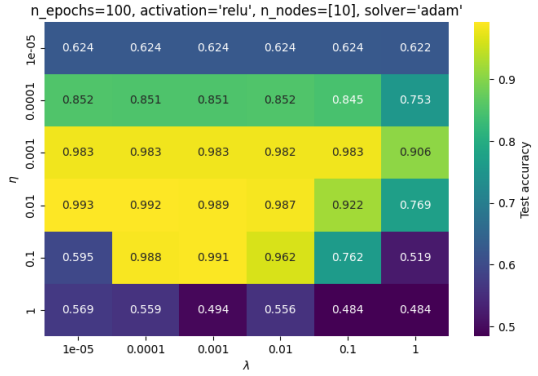

Figure 8: Grid plot finding optimal parameters learning rate η and regularization term λ.

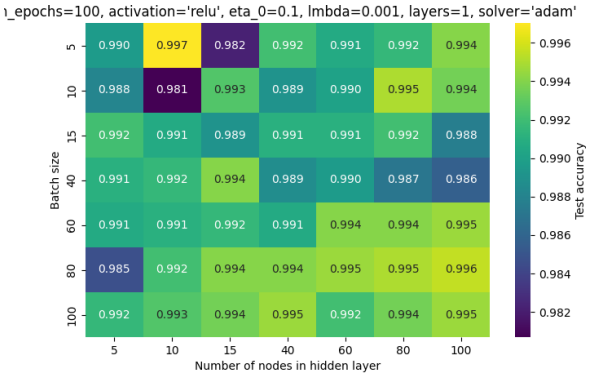

Figure 9: Grid plot finding optimal parameters for batch size in for the stochastic optimizer and number of nodes in the hidden layer.

In the end we can use the different optimal variables to find the best possible prediction rate for our neural network. Using the ReLU activation function, $\eta = 0.01$, $\lambda = 10^{-5}$, n_batches = 5, n_nodes = 10 and n_epochs = 1000 we obtain the accuracy 0.979, and the confusion plot in figure 10.

C. Classification tree

Once more we compare with and without scaling, and obtain accuracy 0.937 and 0.950 using the Gini criterion. Here we also tried the two other criterions Entropy and Log_loss, which have the same accuracy score of 0.945. Using the Gini index and scaling we then plot accuracy score with respect to tree depth in figure 11, and find the optimal tree depth to be 30. Using this we obtain

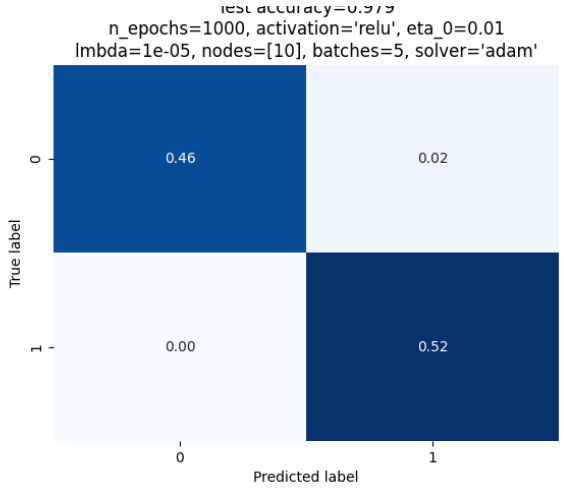

Figure 10: Resulting confusion matrix for our model showing true/false predictions that were actually true/-false.

a accuracy score of 0.984, and the confusion matrix in figure 12.

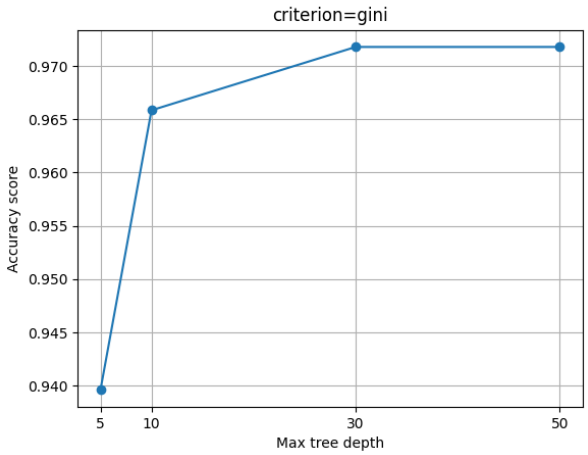

Figure 11: ...

D. XGBoost classification tree

Using XGBoost with depth = 5 and 100 boosting rounds we obtain an accuracy of 1.0 for unscaled data. To look for if the accuracy changes, we then make the grid map in figure 13, before using the chosen optimal values of depth = 5 and boosting rounds = 30 to obtain an accuracy of 1.0 and the confusion matrix in figure 14.

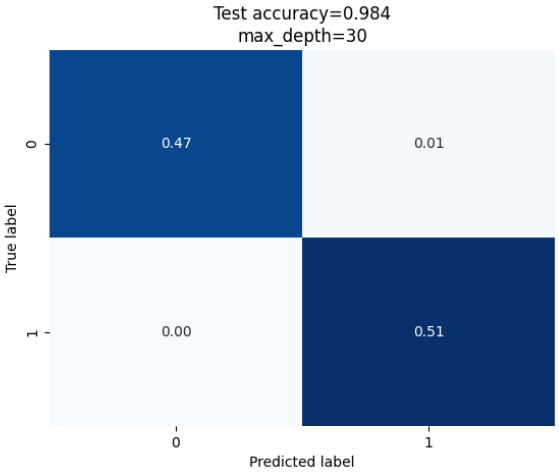

Figure 12: Confusion matrix of the

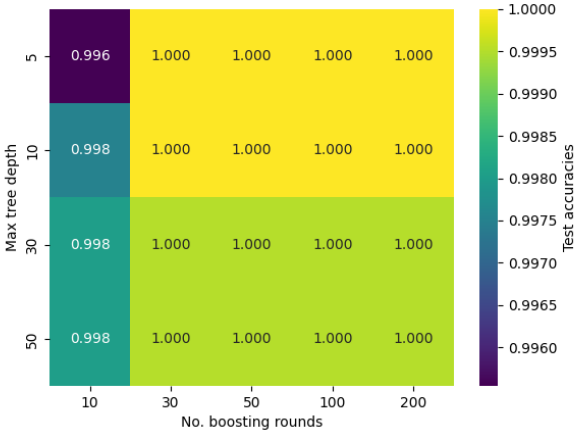

Figure 13: ...

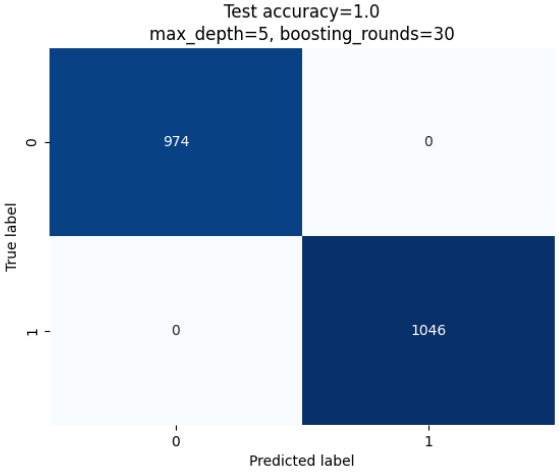

Figure 14: Confusion matrix of the

V. DISCUSSION

A. Logistic regression

B. Classification neural network

C. Classification tree

D. XGBoost classification tree

VI. CONCLUSION

Appendix A. Github repository

https://github.com/LassePladsen/FYS-STK3155-projects/tree/main/project3

References

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002, June). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*, 321–357. Retrieved from https://doi.org/10.1613/jair.953 doi: 10.1613/jair.953

Dash, S. (2023, November). Decision trees explained — entropy, information gain, gini index, ccp pruning. *Medium*. Retrieved from https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c

Dorsaf, S. (2021). *Comprehensive synthesis of the main activation functions pros and cons.* Retrieved 2023-11-29, from https://medium.com/analytics-vidhya/comprehensive-synthesis-of-the-main-activation-functions-pros-and-cons-dab105fe4b3b

Dubey, S. R., Singh, S. K., & Chaudhuri, B. B. (2021). A comprehensive survey and performance analysis of activation functions in deep learning. *CoRR*, *abs/2109.14545*. Retrieved from https://arxiv.org/abs/2109.14545

GeeksforGeeks. (2023, February). *Activation functions in neural networks.* Retrieved from https://www.geeksforgeeks.org/activation-functions-neural-networks/

Hjorth-Jensen, M. (2023). *Applied data analysis and machine learning.* https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html. ([Online; accessed 30-September-2023])

IBM. (n.d.). Retrieved from https://www.ibm.com/topics/decision-trees

Pladsen, L., Qanbari, P., & Vattøy, S. V. (2023). Github - fys-stk3110. *GitHub*. Retrieved from https://github.com/LassePladsen/FYS-STK3155-projects/blob/main/project2/results/FYS_STK3155_Project2.pdf

T, S. (2021, December). Entropy: How decision trees make decisions - towards data science. *Medium*. Retrieved from https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8

Wilson, P. A. (2017, January). *The exoplanet transit method — paulanthonywilson.com.* Retrieved from https://www.paulanthonywilson.com/exoplanets/exoplanet-detection-techniques/the-exoplanet-transit-method/

Winterdelta. (2017, April). *Dataset.* Retrieved from https://www.kaggle.com/datasets/keplersmachines/kepler-labelled-time-series-data/

Winterdelta. (2019). *Github - winterdelta/keplerai: Machine learning project to discover exoplanets.* Retrieved from https://github.com/winterdelta/KeplerAI