FYS-STK3155 Project 3

Lasse Pladsen, Parham Qanbari, & Sander V. Vattøy

December 2, 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 5657 stars in which only 42 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 3197 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 stars 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

$$\bar{x}_{:j} = \frac{1}{n_{rows}} \sum_{i}^{n_{rows}} x_{ij}$$

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

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 which shows the four major prediction scores: the false negative, true negative, false positive, and true positive. In this sense a negative prediction is one that predicts the class 0 (no exoplanet), and positive is the prediction of 1 (exoplanet). The confusion matrix is defined as

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

where each of the metrics are defined as below:

$$TN = \text{true negative} = \frac{\text{\# of correct negative predictions}}{\text{Total \# of negative predictions}}$$
$$FN = \text{false negative} = \frac{\text{\# of false negative predictions}}{\text{Total \# of negative predictions}}$$
$$TP = \text{true positive} = \frac{\text{\# of correct positive predictions}}{\text{Total \# of positive predictions}}$$
$$FP = \text{false positive} = \frac{\text{\# of false positive predictions}}{\text{Total \# of positive predictions}}$$

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,

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

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 (equation 7) (Pladsen et al., 2023).

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

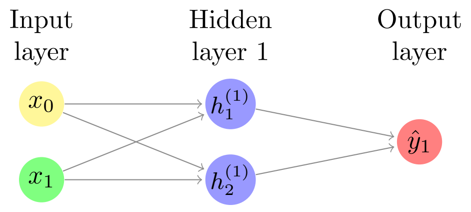


Figure 1: Neural network schematic.

The output of the function $\mathbf{z}_l$ in the hidden layer is passed through an activation function resulting in a function (equation 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 neural signalling is dependent on the activation level: a neuron does not fire upon any activation, it requires a certain threshold before firing (Hjorth-Jensen, 2023). Activation functions in neural networks are set perform in a similar way. They add 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).

The ReLU (Rectified linear unit) function (figure 2) is one often used for activation and is defined as

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

This activation provides an output that ranges from 0 to infinity

$$f(x) \in [0, \infty] \tag{10}$$

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.

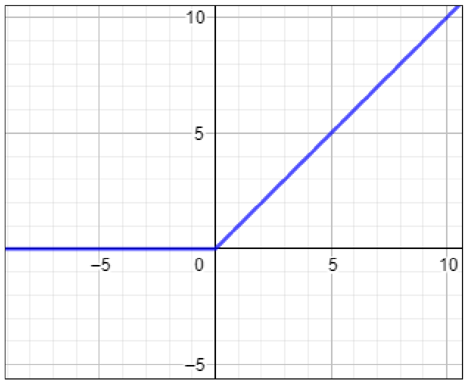


Figure 2: The ReLU function.

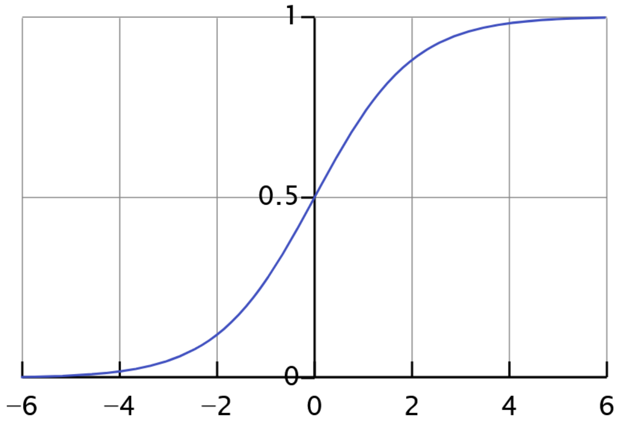


Figure 3: The Sigmoid logistic function, from wikipedia.

The Sigmoid function (equation (equation 5)) is another widely used activation function, see its graph in figure 3. 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.

The hyperbolic tangent function (tanh) is another one used for activation, it is plotted in figure 4 and is defined as

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

Its output has the range

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

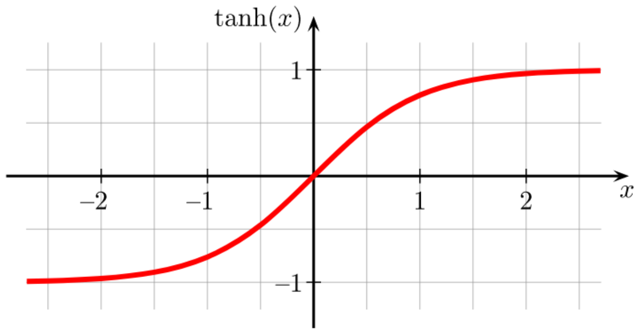


Figure 4: The hyperbolic tangent function, from wikipedia.

The final activation function we wish to compare is no activation at all, that is to say

$$f(x) = x$$

this is often called the *Identity* activation function in relation to neural network code.

E2. Backpropagation

We use the algorithm known as backpropagation to train our feed-forward neural network. For a network we want to optimize the weights **W** and the biases **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 tree

A classification decision tree, or just a classification tree, is a machine learning method that creates a binary tree structure of conditions to classify the input as one of the output classes.

As illustrated in figure (5) such a tree starts with a root node. Each node branches into two new nodes each, and the output nodes are called leaf nodes or just leaves. The middle nodes that are not a root node or a leaf are called internal nodes (Hjorth-Jensen, 2023; IBM, n.d.). The root node and internal nodes each contain a logical condition to classify which branch direction the data should continue in. The leaf represents an output class, which in our case it would be either ”no exoplanet” or ”exoplanet”.

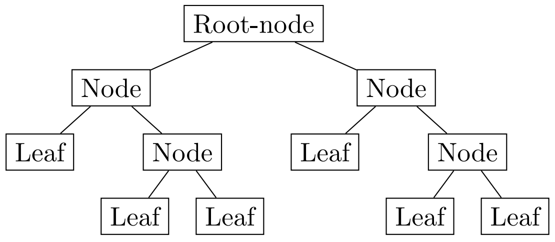


Figure 5: Schematic of a decision tree structure

Decision trees are prone to overfitting, therefore it is common to set a threshold for how deep the tree goes (IBM, n.d.). In general the node conditions can be yes/no, however for a data set such as ours with numerical flux observation, the conditions would be numeric.

F1. Node split criterion

The two most common methods of evaluating each node branching/splitting quality are the Gini impurity/index (equation 1) and the information entropy (equation 1) (Hjorth-Jensen, 2023). They both are based on the following probability distribution function

$$p_k = \frac{1}{n}\sum_{i=1}^{n} I(y_i = k)$$

where p_k represents the number of observations of a given class $k = 0, 1, 2, ...K$ from n total observations. For our binary classification problem we would have $k = 0, 1$ ($K = 1$).

The Gini index , also called the Gini impurity, measures the probability for a random instance being misclassified when chosen randomly (Dash, 2023). It is defined as (equation 1):

$$g = \sum_{k=1}^{K} p_k(1 - p_k)$$

The information entropy is defined as

$$s = -\sum_{k=1}^{K} p_k log p_k$$

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 on thermodynamic entropy from physics and is suited well for decision trees, because it bases the splits on whether or not it causes a information gain (a more pure classification) (Dash, 2023).

F2. 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.

F3. Pseudo-algorithm [??]

[SKAL VI HA MED ALGORITME...? KAN HA EN SÅNN KORT NØKKELORD-FRAMGANGSMÅTE?]

G. Gradient tree boosting

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

III. METHODS

As mentioned our star dataset (section 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 two 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 out the number of occurrences of each class. We chose to do this to just the training set and then re-split that into a new training and testing set. One could also include the original testing data by combining the data before doing this preprocessing. We use Scikit-learn's `sklearn.model_selection.train_test_split` with a testing size of 20% for our split.

For each of the different classification methods we try out scaling the data vs not scaling before training, to see if it improves performance or not. We use Scikit-learn's standardization implementation `sklearn.preprocessing.StandardScaler`.

We use the accuracy (equation 2) to evaluate the prediction performances using Scikit-learn's `sklearn.metrics.accuracy_score`. In addition, we plot the confusion matrices (equation **??** 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 then 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 each hidden layer n_{nodes}. We can write the last two as a merged vector where each element is the number of nodes for that hidden layer:

$$\mathbf{n}_{nodes} = (n_{\text{nodes, layer 1}}, n_{\text{nodes, layer 2}}, \dots)$$

The number of nodes for the input layer is the number of features in the training set, which is 3197, and for the output layer we only have one classification label (either 0 or 1) so there is only one node.

Initially to test if scaling improves performance we use the following parameters and architecture

$$\begin{aligned}
\lambda &= 0.001 \\
\eta_0 &= 0.1 \\
n_{batches} &= \text{auto (Scikit-learn)} \\
\text{activation} &= \text{ReLU (equation 9)} \\
n_{epochs} &= 100 \\
\mathbf{n}_{nodes} &= (10)
\end{aligned}$$

Afterwards we try out different numbers of hidden layers in our network. Then, we test the activation functions other than ReLU, which is: Sigmoid (equation 5), Identity (equation 1), and finally tanh (equation 11). Furthermore, we do a grid search plotting accuracy to η_0 and λ. Lastly, we do a similar grid search for $n_{batches}$ and $\mathbf{n}_{nodes}$ 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 (equation 1). Afterwards we test the information entropy (equation 1) method as the criterion.

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

For the chosen initial parameters, we found that scaling does improves the accuracy:

$$\begin{aligned}
\text{No scaling}: &\ 0.899 \\
\text{Scaling}: &\ 0.979
\end{aligned}$$

We then plot the accuracy with respect to the L2 regularization λ shown in figure 6, we found the optimal value to be

$$\lambda = 10^{-5}$$

Making our final prediction we conclude the accuracy of the logistic regression method to be

$$\text{Accuracy } = 0.981$$

with the confusion matrix plotted in figure 7.

B. Classification neural network

For the chosen starting configuration (see methods B) we found that not scaling the data proved slightly better, the accuracies with ReLU activation and $\mathbf{n}_{nodes} = (10)$ were

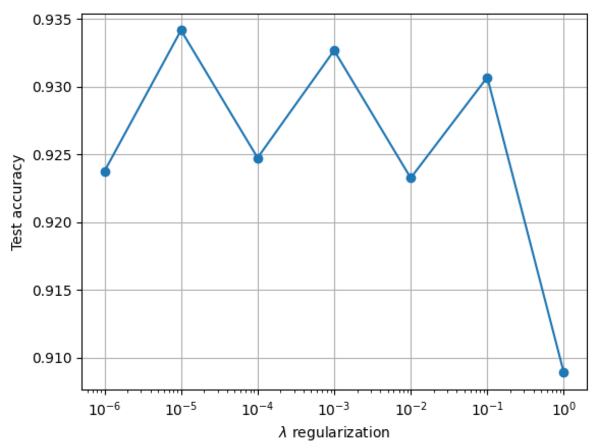


Figure 6: Logistic regression accuracy plotted as a function of the L2 regularization λ.

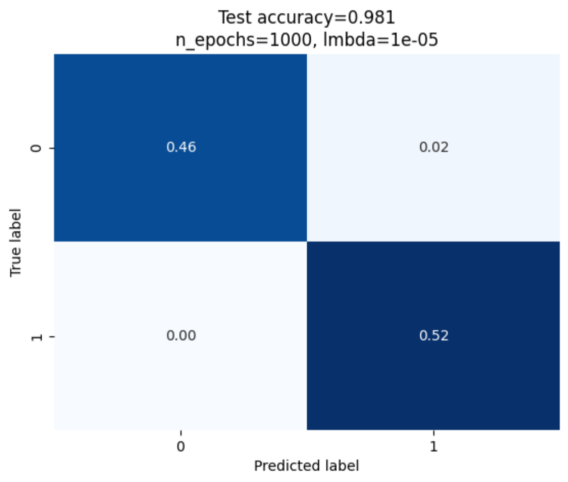


Figure 7: Confusion matrix of the logistic regression's final prediction.

$$\text{No scaling}: \ 0.980$$
$$\text{Scaling}: \ 0.969$$

We then found increasing the number hidden layer from one decreased this accuracy (without scaling) to

$$\mathbf{n}_{nodes} = (10, 10, 10): \ 0.482$$

We then tried out the different activation functions other than ReLU and found with $\mathbf{n}_{nodes} = (10)$ the following accuracies

$$\text{Identity}: 0.746$$
$$\text{Sigmoid}: 0.808$$
$$\text{Tanh}: 0.520$$

meaning ReLU showed the best performance.

The grid search for optimal η_0 and λ values is plotted in figure 8 where we found the optimal

$$\eta_0 = 0.01$$
$$\lambda = 10^{-5}$$

The final grid search to find the optimal number of batches and nodes is plotted in figure 9 where we found the optimal values as

$$n_{batches} = 5$$
$$\mathbf{n}_{nodes} = (10) \ \text{(one layer with 10 nodes)}$$

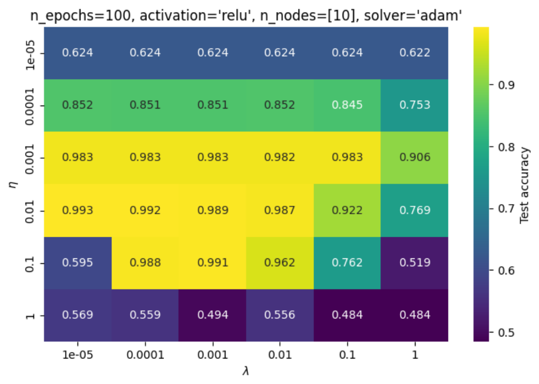
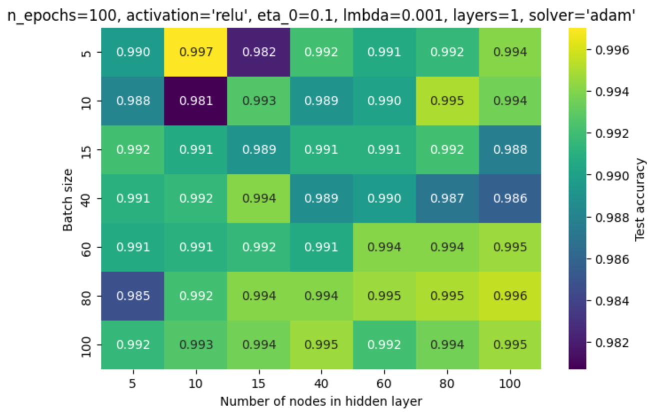


Figure 8: Neural network grid search to find the optimal values for the parameters initial learning rate η_0 and L2 regularization term λ.

Using these found optimal parameters our final prediction for this neural network method proved to be

$$\text{Accuracy} \ = 0.979$$

The confusion matrix is plotted in figure 10.



Batch size \ Nodes	5	10	15	40	60	80	100
5	0.990	0.997	0.982	0.992	0.991	0.992	0.994
10	0.988	0.981	0.993	0.989	0.990	0.995	0.994
15	0.992	0.991	0.989	0.991	0.991	0.992	0.988
40	0.991	0.992	0.994	0.989	0.990	0.987	0.986
60	0.991	0.991	0.992	0.991	0.994	0.994	0.995
80	0.985	0.992	0.994	0.994	0.995	0.995	0.996
100	0.992	0.993	0.994	0.995	0.992	0.994	0.995

Figure 9: Neural network grid search to find the optimal values for the parameters batch size $n_{batches}$ and number of nodes $\mathbf{n}_{nodes}$ in the one hidden layer.

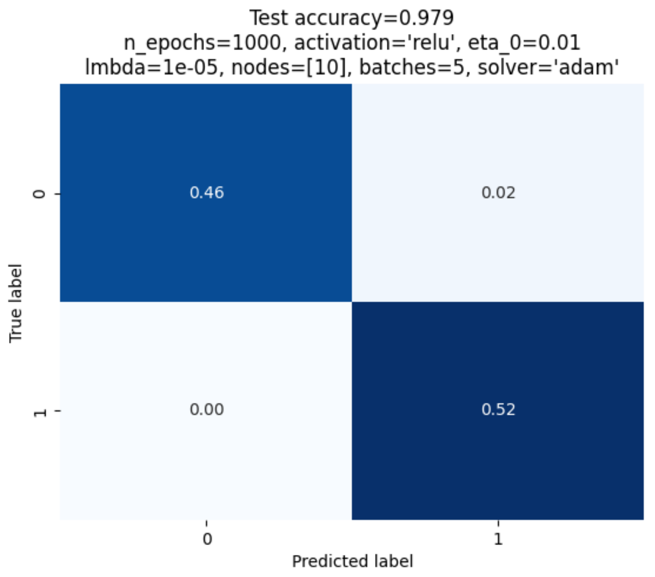


Figure 10: Confusion matrix of the neural network's final prediction.

C. Classification tree

For the classification (decision) tree we start using the Gini index with a max tree depth of five which results in the accuracies

$$\text{No scaling}: \ 0.940$$
$$\text{Scaling}: \ 0.939$$

showing that in this case there is barely any difference when scaling. We move on without scaling replacing the criterion with entropy (equation 1) which showed slightly worse performance:

$$\text{Entropy}: \ 0.934$$

Using the Gini index with no scaling we find the optimal maximum depth considering both computational time and performance to be 30, the analysis is plotted in figure 11.

Using this configuration we obtain the final prediction accuracy

$$\text{Accuracy} \ = 0.984$$

where the confusion matrix in plotted in figure 12.

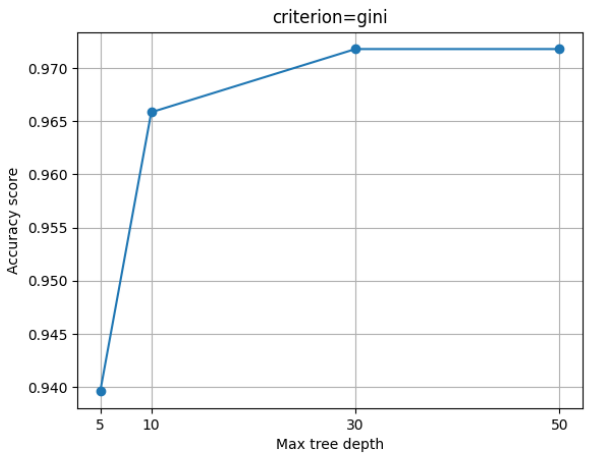


Figure 11: The classification tree accuracy plotted as a function of the maximum tree depth. Anything more than 30 depth proved to not increase the accuracy. It is possible that the phenomena of overfitting can be shown by extending the x-axis, however the computational time is already huge for these values.

D. Gradient boosted classification tree

Using the XGBoost classification tree with depth $= 5$ and 100 boosting rounds we immediately obtained the perfect accuracy of

$$\text{Accuracy} \ = 1.000$$

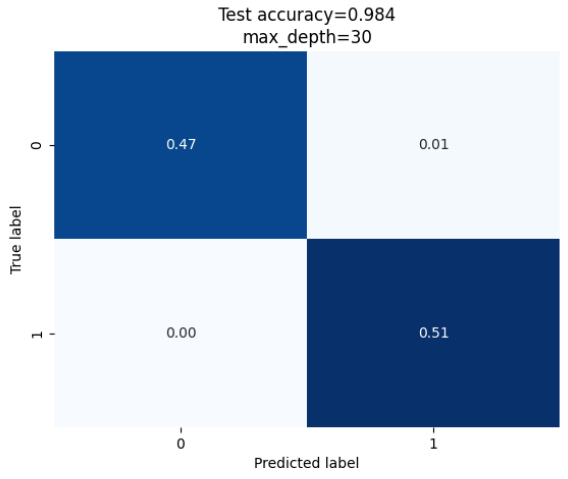
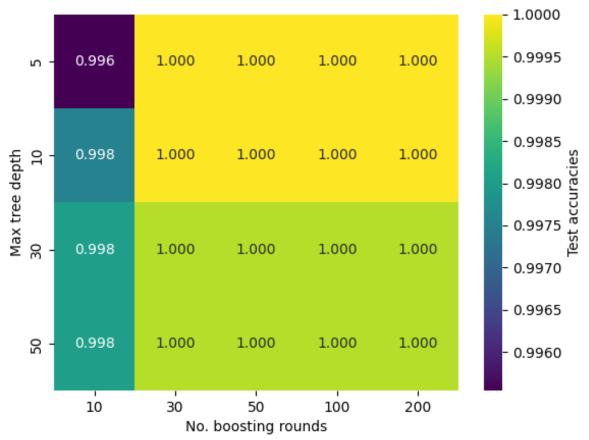


Figure 12: Confusion matrix of the classification tree's final prediction.

We still plotted the grid search to look for a change in accuracy which is plotted in figure 13 which showed we could reduce our number of boosting rounds down to 30 while keeping the perfect accuracy. We found no further reason to keep tweaking our XGBoost configuration since we already had a perfect prediction, the confusion matrix is plotted in figure 14.



Max tree depth \ No. boosting rounds	10	30	50	100	200
5	0.996	1.000	1.000	1.000	1.000
10	0.998	1.000	1.000	1.000	1.000
30	0.998	1.000	1.000	1.000	1.000
50	0.998	1.000	1.000	1.000	1.000

Figure 13: XGBoost tree grid search to find the optimal number of max tree depth and number of boosting rounds.

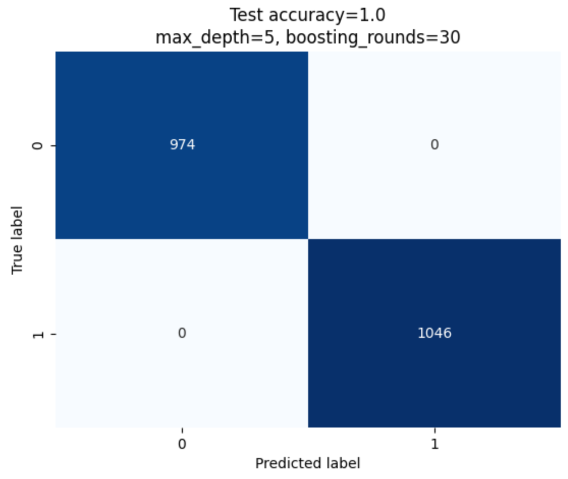


Figure 14: Confusion matrix of the XGBoosted tree's final prediction.

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

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