TITTEL

Arne Elias Tidemand Ruud & Bjørn Magnus Hoddevik
University of Oslo

(Dated: December 23, 2022)

In this report we study a COVID-19 dataset provided by the Mexican government. It has over a million entries, but after preprocessing the data we narrow it down to around 300 000 entries. Based on if the patient died, was intubated or submitted to the ICU, we created the high risk category to use as our target. Other input categories are for example what previous illnesses the patient have, their sex and their age. We discuss how different metrics can be used to determine if a model is good or not and why this is important. In the article we look specifically at logistical regression and a categorical neural network. We compare the accuracy metric to Matthews correlation coefficient (MCC) while also looking at the confusion matrix. After studying how they differ when we look at a balanced dataset and not we find that none of the metrics can fully replace the confusion matrix. We find that accuracy is not indicative of a good model even with a balanced dataset. When comparing a model trained on a balanced dataset and then tested on both a balanced and unbalanced test set, we find that even though the first, $MCC = 69.3\%$, gives a slightly better MCC score but does worse at correctly identifying high risk patients, 77.6% compared to 27%, than the latter, $MCC = 67.6\%$. Therefore, the confusion matrix remains relevant. We then attempt to use logistic regression and decision trees. Resulting in MCC values of 0.65 and 0.66 respectively.

I. INTRODUCTION

In this project we wish to study how impactful preprocessing can be when training a categorical neural network. When training your network, you can end up with a good accuracy score that is close to 100%, however this does not mean your network is good at prediction. One such cause, which we will study here, is an unbalanced dataset. When the ratio between true and false becomes as extreme as 1:9, a 90% accuracy most likely means you have trained your network to always guess the more common of the two. This is something you can avoid by using correct metrics to get a better understanding of your accuracy and by studying the confusion matrix to see what your model predicts. This information can then be used to reconsider what biases your data may have.

We will discuss briefly how we treated the raw data to better match our model. This includes getting rid of useless information, reformatting data and defining our target. A good understanding of what your data tells you is important when finding bias in your model. We wish to have our input and target data in binary form, meaning some data must be converted to binary and others to one-hot vectors. Then attempt to use some different machine learning methods: including neural network, logistic regression and decision trees to see how the different methods performs on the processed data.

II. THE DATASET

The dataset we will be studying is a COVID-19 dataset provided by the Mexican government with over one million patients. Most categories are given a value of 1 or 2 meaning yes or no respectively. However, if there is data missing they are given a value of 97 or 99. There are a few categories which deviate from this; the date died category has dates, where a date 9999-99-99 is given when the person did not die, medical unit category has several values 13 unique values, age is given in years and classification final has 7 unique values where a value higher than 3 means the covid test was negative.

There are also some categories that require extra attention, such as the pregnancy category. Only women who were issued a pregnancy test were classified as 1 or 2 with the rest having no data, meaning if we were to remove patients which are misssing it we would also remove all men. Therefore, we give all patients with no data a value of 2 in this case. Similiarly, we assumed for the ICU and intubed categories that all missing entries meant no (2).

As for age, we decided to divide into age groups. This is common practice within such statistics and will allow us to only have binary inputs. Doing it this way also groups certain periods of life together which also have similiar health issues which might help the network learn faster. We decided to use the age groups 0-17, 18-29, 30-39, 40-49, 50-64, 65-74, 75-84, 85-120.

Some categories missing data also meant these entries were not usefull, unless some assumptions were made. We wish to keep the data as close to relaistic as possible and did therefore not make any further assumptions than mentioned above. Meaning, we removed some patients which missed data in all categories except: Age, classification final, ICU, intubed and pregnant.

For the classification final category we decided to drop all patients with a classification higher than 3. This is under the assumption that no patient who tested positive would be considered high risk of dying. Having done so, we could then get rid of the entire category, as it no longer provided any usefull input.

Lastly, we had to determine what would be considered our target. We want to be able to judge based on data a doctor can immediately find out at admittance. This meant categories such as ICU, intubed and date of death can not be included in inputs. Having one of these categories as positive would also mean the patient would be high risk for serious complications and death, and were also a prefect target. Therefor, we made a new category called high risk which was assigned a value 1 if any of the previosly mentioned categories had a value of 1. Afterwards we removed these categories, only keeping the high risk category.

Having done all this, we were left with categories only consisting of yes or no data. Originally, yes had the value 1 and no the value 2; to keep in line with common boolean values, 2 was changed to 0.

A subject of further discussion for optimization would be to refer to a correlation matrix, and removing categories with little to no correlation to high risk. In our case however, we decided to not do so.

III. METHOD

A. Categorical neural network

We wish to have a very flexible network, where we can easily configure parameters and change our model in search of better accuracy. We wish to study how the structure of the neural network makes a difference and how hyperparameters matter. We therefor do a grid search to find optimal complexity, learning rate and regularization parameter. To perform this grid search in the hunt for the most optimal parameters, our program must accept a parameter which contains information about our grid search. We have chosen to do this by providing a nested list, which contains four lists; one for each parameter. These lists provide the name of the parameter and a range of values we want to try for two parameters and the other two must be single elements.

Although the grid search feature makes using the program a bit easier, there are a lot of hidden requirements. As mentioned, only two parameters can be used for a grid search at a time. Additionally, the learning rate parameters is not used for all optimizers, such as ADAM. This brings us to a second feature our program must have; the program must accept different optimizers. It will however be up to the user to be aware of what the different optimizers require.

Before we can find optimal parameters however, we must first study different metrics. Metrics such as accuracy is very straight forward, and lets us know how often we are correct. This however is not always connected to how good our model is. As previously mentioned, our dataset contains a small minority of high risk patients, and a model which never guesses high risk will consequently have high accuracy without being helpful towards our goal. Instead we must look at different metrics which may give us a better picture.

One such metric is the Matthews correlation coefficient (MCC) (Chicco, 2020)[1]. It uses equation 1 to evaluate the model. A model which guesses correctly always would get a score of 1 and -1 if it guesses wrong always. If we refer to the mcc being 100% it means +1. If its -1, it would be negative 100%, meaning they are directly opposite correlated. A model which tries to predict coinflips would then have a score of 0. Supposedly this metric only gives a high score if the model predicts right the majority of the positive cases and negative cases. So, if our model only guesses one option, it will result in a lower score. This proves beneficial when a dataset in where the outcomes is heavily skewed towards one outcome. In which case the use of always of accuracy might be misleading as it would perform well just constantly predicting the one with the highest outcome everytime.

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)\cdot(TP+FN)\cdot(TN+FP)\cdot(TN+FN)}} \tag{1}$$

The MCC metric essentially gives us a recap of what we might expect from the confusion matrix. So while MCC is perfect for our dataset, being able to produce a confusion matrix is still useful for finding out what our model ends up predicting. So an option to produce a confusion matrix should also be a feature.

When first training our network, we must balance our dataset. This means one of two things, resampling the minority or downsampling the majority. Since we have no shortage of datapoints, we chose the latter. We perform this downsampling by finding the amount of datapoints which belong to the minority and then randomly choose that many datapoints out of the majority without replacement.

After having found the right complexity and hyper parameters, we can test our model on an unbalanced dataset and measure its performance using MCC. As an extra reinsurance, we will also study the confusion matrix of that prediction.

B. Logistic Regression

Logistic regression is a supervised learning algorithm used for classification tasks. It is a type of regression analysis that is used to predict a binary outcomes. As in either 0/false or 1/true. In logistic regression, the output is a probability that an input belongs to the specific class in question. The predicted probability is transformed into a binary value (0 or 1) by a threshold function, which is typically set to 0.5. Also often referenced as a stepwise function.

$$f(x) = \begin{cases} 0 & x \leq 0.5 \\ 1 & x > 0.5 \end{cases}$$

If the predicted probability is above the threshold, the input is classified as belonging to this class, and if it is below the threshold, it is classified as belonging to the other class. In this dataset, 0 meaning low risk and 1 means the person belongs in the high risk group. The goal of logistic regression is to find the best weights for the input features that will result in the highest accuracy. This is done by minimizing the error between the predicted probabilities and the true labels using an optimization algorithm, such as gradient descent. Using sklearn built in functionality the default option using solver by name of Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm. In short it is an close to the Newton's Method, except the Hessian matrix is approximated using specified by gradient evaluations (or approximate gradient evaluations). Using estimation to the inverse Hessian matrix. Additionally it only stores a few vectors that represent the approximation implicitly. [3] For a deeper dive we refer article written by Aria Haghighi, data science and engineering leader at Amperity, Facebook, and Apple. [2] Additionally l2 penalty is applied. The penalty is set to (w = weights)

$$\ell_2 = \frac{1}{2}\|w\|_2^2 = \frac{1}{2}w^T w$$

Logistic regression can be used for binary classification tasks as well as multi-class classification tasks (using a one-vs-rest approach). It is a simple and effective algorithm that is widely used in many different fields, including finance, medicine, and marketing. Meaning its a method which is natural to test out.

C. Decision trees

In a wish to explore a very different method, we chose to explore shortly about decision trees. A method when compared to models such as neural networks is seemingly much less complex. With fewer parameters to tune and are easier to understand and interpret this supervised learning algorithm is quite useful in many cases. It can be used both for classification and regression tasks where its goal its either to predict a value or a label. For a simple tree the structure can look like this 1 The learning algorithm builds a tree by starting at the root node and splitting the data based on the most important feature that maximizes the information gain at each step. The process continues until the tree is fully grown or until the information gain becomes zero or it has reached its max samples per node, which means that the tree can't be split any further. This method is also known as CART (classification and regression trees). Once the tree is built, a new data sample is uses the tree starting at the root and moves along one branch at a time, stopping once it reaches a leaf node which corresponds to a certain label or value depending on the problem.

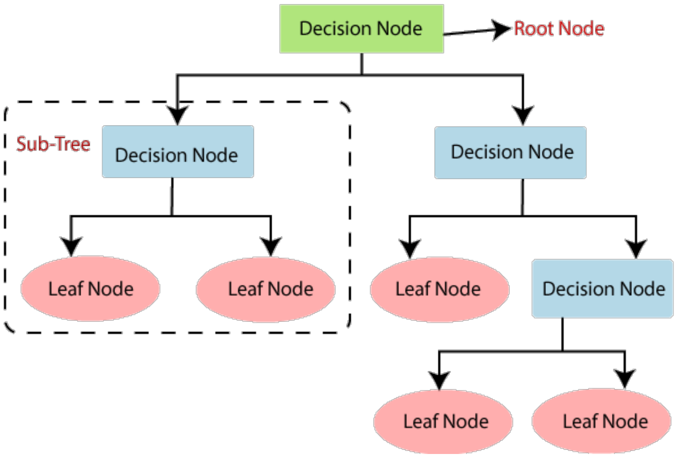

Figure 1. Simple decision tree structure with one root node and five leaf nodes.

Using sklearn built in method we do a gridsearch trying different parameters for criterion (gini, entropy), max depth, min samples split and min samples leaf. [4] Criterion is the function to measure the quality of a split. Gini and entropy both measure the impurity of the node which it then uses to decide how to split the node. Entropy is defined as

$$H(x) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i).$$

Shannon entropy, measure of the amount of uncertainty or randomness in a given set of data. p(x) is probability of the specific x event (2)

The gini impurity is calculated with this equation:[5]

$$Gini(x) = 1 - \sum_{i=1}^{n} p(x_i)^2.$$

Gini impurity. Measures the frequency at which any element of the dataset will be mislabelled when it is randomly labeled. (3)

The other parameters such as max depth of the tree determines how many branches the model is able to create. Limited the amount of branches can hinder potential overfitting which is common in the use of decision trees. For more documenting of all the parameters we refer to official scikit learn documentation. [4]

IV. RESULTS

A. Categorical neural network

Without a balanced dataset, we get the grid search in figure 2 for learning rate $\eta = 0.001, 0.01, 0.1, 1, 10, 100$ and regularization parameter $\lambda = 0.001, 0.01, 0.1, 1, 10, 100$.

Next, we choose the best combination and do a similiar grid search for complexity with $n_{layers} = 2, 3, 4, 5$ and $n_{neurons} = 10, 30, 100$ and get figure 4.

This does not give us much information, so we look at similiar searches using MCC as a metric. Figure 3 shows these results.

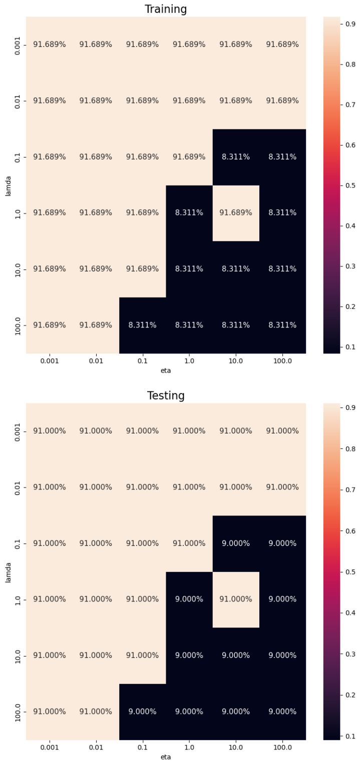

Figure 2. A grid search for best hyper parameters using accuracy on an unbalanced dataset. With learning rate eta along the x-axis and regularization parameter lambda along the x-axis.

What figure 5 tells us is further backed up by the confusion matrix in figure 3.

Having looked at what an unblanaced training set gives us, we move forward to a balanced data set. The grid search for best hyper parameters is shown in accuracy and MCC in figure 6 and figure 7 respectively. Similiarly, figure 8 and figure 9 shows the best combination of $n_{neurons}$ and n_{layers}. Figure 10 shows the new confusion matrix with a MCC score of 69.3%.

Finally, we see in figure 11 the confusion matrix of our model, which was trained on a balanced set, applied to an unbalanced test set. It had a MCC score of 67.6% and a 77.76% accuracy of predicting if a patient was high risk correctly, compared to a 89.93% accuracty of predicting if a patient was low risk correctly.

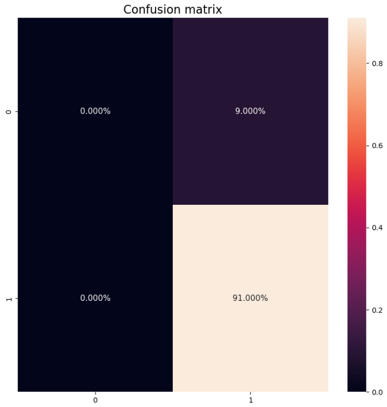

Figure 3. A confusion matrix of a model trained on an unbalanced trainset with $\eta = 0.1$ and $\lambda = 0.01$, $n_{layers} = 2$ and $n_{neurons} = 10$ and tested on an unbalanced trainset.

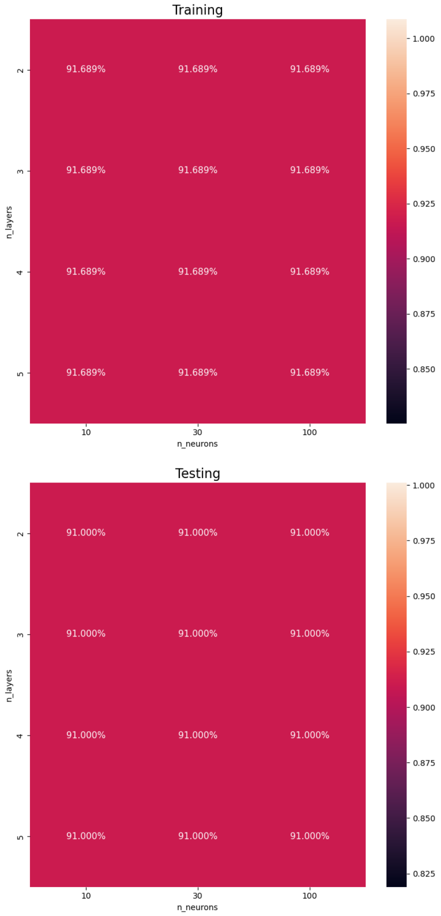

Figure 4. A grid search for best complexity using accuracy on an unbalanced dataset. With number of neurons along the x-axis and number of neurons along the x-axis.

B. Logistic regression and Decision tree

As we make use of built in models there is not as much results as of big gridsearch to present. However we do all the search within the sklearn framework with the use of GridSearchCV function which makes use of cross validation. Which for the decision tree searches and finds the best model in very short time span. For the logistic regression we tested a few different parameters, however the best result was running the default solver(lbfgs) with a l2 penalty. We then calculate accuracy and mcc score for both. For the decision tree the parameters found in the given dictionary we set up was: criterion = "entropy", max depth=7, min samples split=2, min samples leaf=1. Both methods we are using the balanced dataset. For logistic regression we get an accuracy of 82.30% and mcc of 0.65. While we get a slightly better result with decision tree with 82.75% accuracy and an mcc of 0.66. As you can see they perform quite similar, close enough to say anything definitely of which is better as some tuning or better choices/random state might result in one nudging just over the other method. However judging just by mcc we can see they both perform below that of which the neural network managed to produce, which was was of of an mcc equal 0.693.

V. DISCUSSION

In figure 2 we see what might appear to be a good model, no matter what hyper parameters we choose we get a very high accuracy. While it appears the lower right region of the grid searches have a very bad accuracy, it is just the model predicting the opposite of what we want it to. What this tells us is that, by comparing the amount of lighter squares to the darker ones, our model is more likely to lock into guessing only the choice which is a majority. Figure 4 tells us that the complexity of the model has no say in how our model performs.

As previously mentioned however, accuracy is not a good measure of how well our model performs. This becomes clear in figure 3, where we see that the minority option "high risk" is never guessed. Our accuracy is at this point only an overly complex way to tell the percentage of our data set which consists of low risk patients.

A much better evaluation of our model is given in figure 5. Here we see clearly that the model has no idea of how to correctly predict. There is no need for further grid searching for the right complexity, as the point has already been made.

Moving forward, we look at how a balanced train set makes all the difference. In figure 6 we see how hyper parameters make a difference again. Models which are good at predicting have a MCC score, the lower learning rate and regularization parameter have the best scores. With $\eta = 0.1$ and $\lambda = 0.01$ we can expect MCC scores around 69%. Now, with good MCC scores, accuracy becomes relevant again. In figure 7 we see that the best MCC scores in figure 6 roughly corresponds to an accuracy of 84%. An interesting difference from the two figures is the difference in score for $\eta = 0.01$ and $\lambda = 0.1$. With only a MCC score of 35.6%, the same combination has an accuracy of 77.3%, which is a higher accuracy than what the models with $\lambda = 0.01$ and $\lambda = 0.001$ and the same η have. However, these same combinations had a much higher MCC score. So even still, with a balanced train set, the MCC score and accuracy score can tell two vastly different stories. It appears this particular combination of hyper parameters predicted low risk more than the two previoulsy mentioned combinations.

In figure 8 we have the complexity grid search for our model. The different complexity combinations range from 66% to 68% in MCC score, with one outlier being 5 layers and 10 nodes. Why this happens I am unable to figure out, it does not happen all the time, but is a regular occurence. We see that a balance between layers and neurons gives best MCC scores, with 4 layers and 30 neurons being the best. As computation time rises exponentially with complexity, this may not be worth the investment. The least complex combination and the best only differ in 0.2%, and the leas complex model may even prove better in other training rounds. Figure 9 also tells the same story, this time with no outliers.

In figure 10 and figure 11 we proceeded with $\lambda = 0.01$, $\eta = 0.1$, $n_{layers} = 2$ and $n_{neurons} = 10$. We wanted to see how important it is to test your data on an accurate representation of the real world. Having a balanced test set adds a bias to the data. When comparing the two figures, we see that the model performs better on an unbalanced test set than a balanced one. While the low risk predictions are much better on the balanced set, going from 1% to 10%, the difference in high risk prediction surprisingly becomes better. The FP and TP groups switch places, meaning the model's high risk prediction accuracy goes from 27% to 77.6%. Interestingly, the MCC score of figure 10 was higher than figure 11. Meaning, that even the MCC sometimes fails to give us a good measure of our model.

Lastly, this opens up the discussion of wether our grid searches, even using MCC, tell us something usefull. This is certainly the case for a complexity search, as the little variation in results based on complexity is too small. Additionally, a slightly better MCC score does not mean the model is better at predicting if a patient is high risk or not.

VI. CONCLUSION

To conclude, we found that neither accuracy or MCC does as good of a job as a confusion matrix to tell whether your model is good. What makes your model good is very dependent on what you want it to do, and in our case it was to correctly predict if a patient is high risk or not. For both logistic regression and decision trees we get an mcc of 0.65 and 0.66 respecitvely. Meaning not much difference. However the network does outperform it, at least with just mcc in mind with a a score of 0.69. Accuracy fails completely at this for an unbalanced dataset and only improves marginally when used on a balanced one. MCC is useful to tell what your model is doing, especially when combined with a confusion matrix. After testing the model on both a balanced and unbalanced test set, we found that the MCC fails at determining if our model was good at what we wanted it to, which is to correctly predict high risk patients based on information given. We also saw how little help the grid searches were in this case, as a good MCC would not necessarily mean a better model.

A better way to measure the performance of our model would be to tailor a metric specifically for measuring the accuracy of the model on only high risk patients.

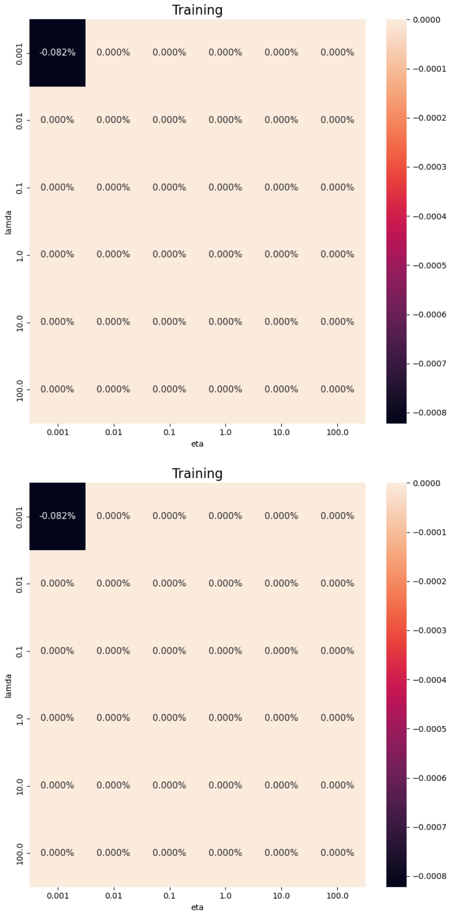

Figure 5. A grid search for best hyper parameters using MCC on an unbalanced dataset. With learning rate eta along the x-axis and regularization parameter lambda along the x-axis.

A. What next?

As we are using quite slow machines, it would be interesting when using the neural networks to see how they would perform over a many times more batches and different parameters to see if there is potential for big improvement. Other potential actions is to further limit the features which are not working as good predictors. Which could reduce the amount of training needed. Other avenues to explore is a continuation of the decision trees, which is random forests. Essentially being a collection of trees, hence the name, which could perform well on a more complex data set. However it does need more rigours training. Therefore needing to further dive into concepts such as bagging and bagging. Lastly finding a way to more easily visualize results such as trees would prove beneficial in presenting results, which would be a big help for both reader and author when testing different paramters and models.

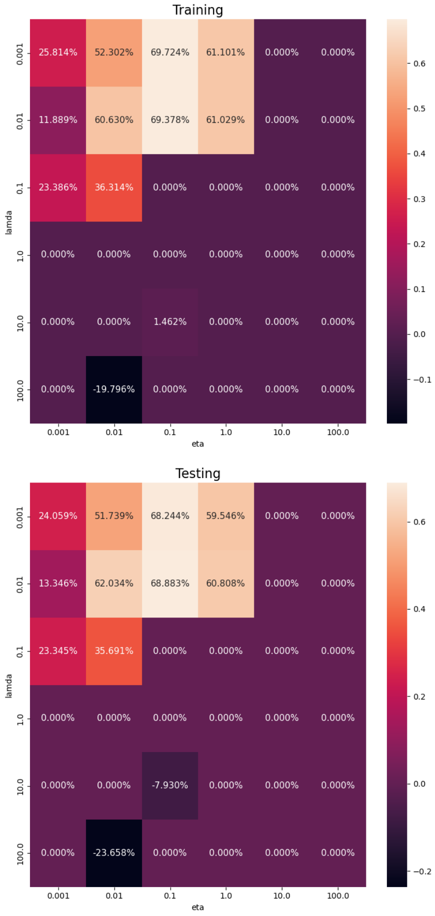

Figure 6. A grid search for best complexity using accuracy on an balanced dataset. With learning rate eta along the x-axis and regularization parameter lambda along the x-axis.

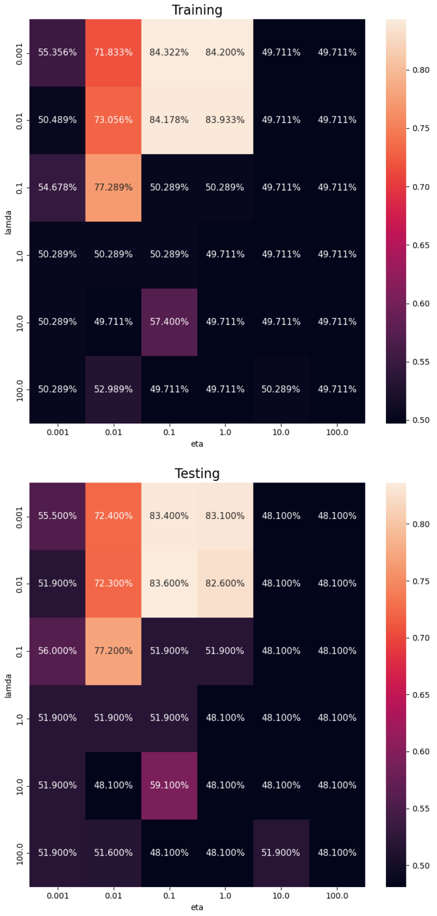

Figure 7. A grid search for best complexity using MCC on an balanced dataset. With learning rate eta along the x-axis and regularization parameter lambda along the x-axis.

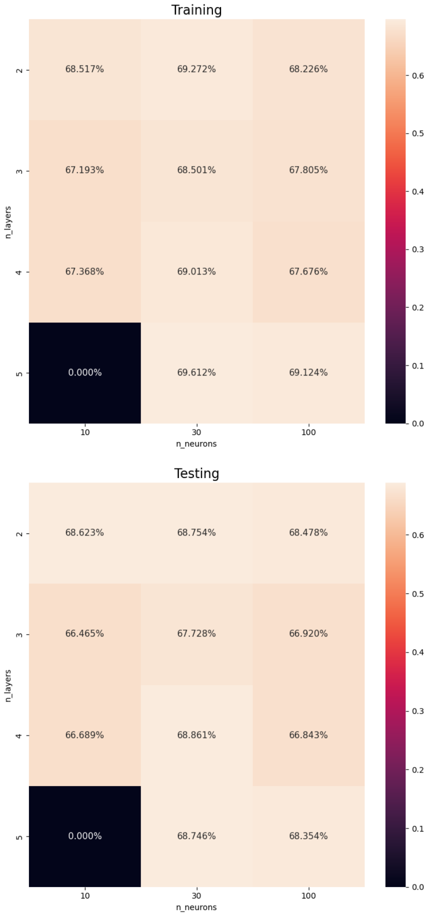

Figure 8. A grid search for best complexity using MCC on an balanced dataset. With number of neurons along the x-axis and number of neurons along the x-axis.

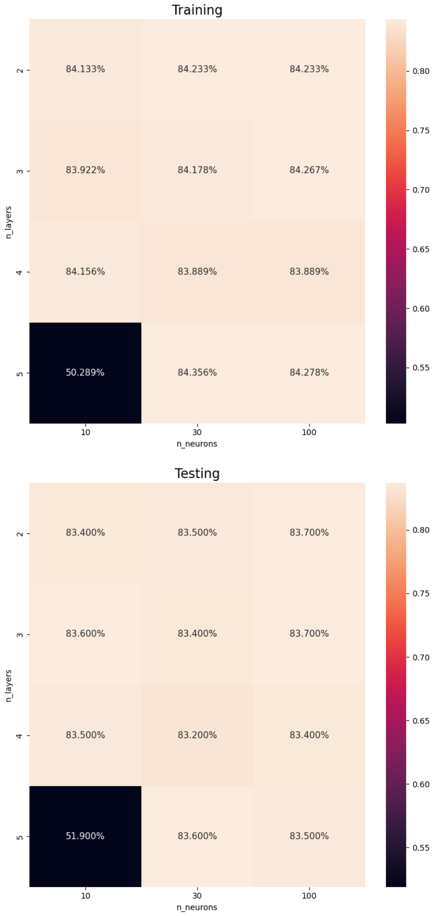

Figure 9. A grid search for best complexity using accuracy on an balanced dataset. With number of neurons along the x-axis and number of neurons along the x-axis.

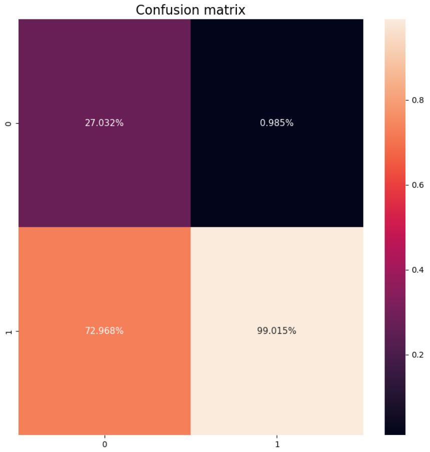

Figure 10. A confusion matrix of a model trained on a balanced trainset with $\eta = 0.1$ and $\lambda = 0.01$, $n_{layers} = 2$ and $n_{neurons} = 10$ and tested on a balanced trainset.

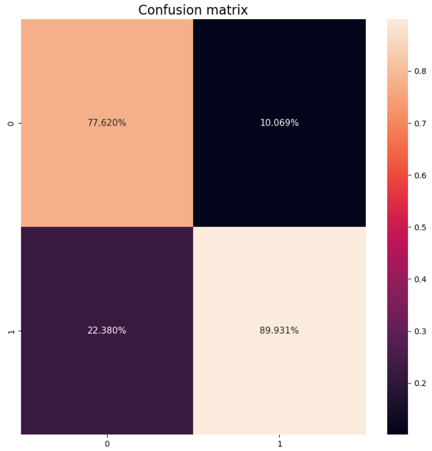

Figure 11. A confusion matrix of a model trained on a balanced trainset with $\eta = 0.1$ and $\lambda = 0.01$, $n_{layers} = 2$ and $n_{neurons} = 10$ and tested on an unbalanced trainset.

[1] Chicco, D., Jurman, G. https://doi.org/10.1186/s12864-019-6413-7 BMC Genomics 21, 6 (2020).

[2] Aria Haghighi, https://aria42.com/blog/2014/12/understanding-lbfgs Article/blogpost (2014).

[3] Aria Haghighi, https://stackoverflow.com/questions/38640109/logistic-regression-python-solvers-definitions StackOverflow (2018).

[4] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. Scikit-learn: Machine Learning in Python. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifie Journal Of Machine Learning Research. (2011)

[5] Pablo Anzar, https://quantdare.com/decision-trees-gini-vs-entropy/ QuantDare (2020).