Predicting a pulsar star using machine learning

M.S Ingstad & F.L Nilsen
(Dated: December 18, 2019)

We have analyzed the dataset from the HRTU2 survey containing candidates for pulsar starts, with the goal of distinguishing real pulsars from the candidates that originate from random noise. We used feed forward neural networks, logistic regression and XGBoost with decision trees, linear classifiers and the DART method. We found that neural networks performed best with an F-score of 0.889 and an area of 0.939 under the PR curve, both evaluated by the k-fold cross-validation error on our dataset. The DART method gave an F-score of 0.886 and area of 0.931 under the PR curve, with significantly lower computation time for convergence than the neural network.

I. INTRODUCTION

A pulsar star is a type of rotating neutron star that produces radio emission out in beams from its poles. When these beams point towards Earth a periodic signal is generated, and so these pulsars can be very useful for astronomers. Thus, classifying stars into pulsars and non-pulsars becomes a important task in astronomy. Although it can be done accurately with human labeling, this is a slow and expensive method for larger datasets. Because of this, finding quick and accurate methods for labeling pulsars is of scientific interest, and is what we will look further into in this report. As the classification of pulsars is a binary classification case, using machine learning classifiers seems to be a good candidate for automatic labeling of pulsars.

In this report we will look at the HRTU2 datasets, with 17,898 pulsar candidates published and analyzed by Lyon et. al [4]. To classify the dataset we will be using neural networks, boosting and bagging of decision trees, linear and dart models and lastly we will use logistic regression. We then compare the results for the the different methods, to see which gives the best results for this classification case, and compare the best results to the results gotten by Lyon et. al.

II. METHOD

Dataset

The pulsar dataset we will be analyzing contains candidates for pulsar stars collected during the High Time Resolution Universe (HRTU) survey. As pulsars rotate, their emission beams hits the Earth in a periodic signal. The data is already picked out from those stars that showed signs of being a pulsar, however more often than not such signals are a product of noise. In this dataset there is then 16,259 non-pulsar stars and 1,639 actual pulsars. All these examples have been checked by human annotators. Each pulsar candidate can be represented by 8 continuous variables with values from the integrated pulse profile in the first 4 predictors and the curve for dispersion measure vs signal to noise ratio (DM-SNR curve) in the last 4 predictors. The values in each set of 4 predictors are statistical properties, respectively the mean, the standard deviation, the excess kurtosis and the skewness. The ninth entry to the data is the target class, which is 0 for non-pulsar stars and 1 for pulsars. [4]. We denote the target class as y, and the 8 predictors as $X0 - X7$, with the first four belonging to the integrated profile, and the last 4 of the DM-SNR curve, with the statistical moments in the same order as listed above.

Classification methods

To classify this dataset we will be using several different methods to analyze the dataset. One of the methods we will be using is a feed forward neural network (NN). This consists of several layers of so called perceptrons, which take all outputs from the previous layers as input, assign a unique weight for all inputs and calculate the sum off all weights times each input, usually there is also a bias term included in the sum. The perceptron then gives an output, given by an activation function evaluated with this sum as the argument. To implement the neural networks in Python, we will be using the library TensorFlow [1].

Another method we will be using is boosting of different classifiers. Boosting combines several weaker classifiers into a stronger one through an iterative process. More specifically we will be using gradient boosting. Given a model $f_m(x_i)$, and a target y_i, gradient boosting finds the next model $f_{m+1}(x_i)$ by training a model on the negative gradient of the cost function $C(y_i, f_m(x_i))$ with regards to $f_m(x_i)$. If this gradient is $h(x)$, the next model f_{m+1} is given by $f_{m+1}(x) = f_m(x) + \nu h(x)$, where μ is the constant that minimizes the loss function for $f_{m+1}(x)$. For gradient boosting we will be using the XGBoost (XGB) package in Python. This package gives several different options for the weaker classifiers, and we will be using the tree method, linear classifiers and the DART methods. XGB also uses bagging for the training. In bagging, one resample the data with the bootstrap method multiple times in order to create several models. One then gets the final output by combining all models. For the tree model XGB also gives the possibility of returning the importance score of the different features for the model, and plotting one of the trees from the bagging

algorithm.

For the tree method, XGB uses multiple decision trees to create what is known as a random forest. For a single tree, the output is decided by going through several nodes. For each node, the tree has a test, and whether or not the test passed determines what nodes the tree goes to next. Each outcome represent a different branch of the tree. At the end of the tree there is an end node, known as a leaf. This then gives the output of the individual tree.

Lastly, we will compare all methods with logistic regression to classify the dataset. Here we calculate t, given by the sum of all predictors times some weight β_i including a bias term β_0. We then calculate the probability of the positive outcome as the sigmoid funciton evaluated at t, so $e^t/(1+e^t)$. To implement this in Python we will be using the method `linear_model.LogisticRegression` from Scikit-learn [5].

In general, all models discussed gives a probability for the positive case x, (and negative case since this is 1 - x). To turn this into a classification we need to set a threshold $a \in [0, 1]$, such that if $x > a$ we predict 1, and if $x < a$ we predict 0. A natural value for this threshold might be 0.5, however this could be changed, depending on the results from the model and the wanted traits of the prediction.

Performance analysis

To measure the performance of the model we will be looking at multiple measures. Since the dataset is fairly unbalanced, one useful metric is the precision-recall curve (PR-curve). This curve has the precision of the model on the y axis and the recall (or true positive rate) on the x axis. The precision is given by the number of true positives over the number of false positives and true positives, while the recall is the number of true positives over the number of false negatives and true positives. Said in other words the PR-curve plots how many of the positive predictions are actually positive vs how many of the actually positive cases gets predicted as such. Because of this, the PR-curve is particularly suited for datasets where the negative case is highly over represented, as it is not affected by the true negatives, which for an imbalanced dataset could be close to the entire dataset. An unskilled classifier (assigns random number to all cases) should get a precision equal the share of positive cases in the dataset, while the true positive ratio should be equal to the threshold, so the line for random guessing should in our case be a flat line with a precision around 0.09. A perfect classifier should then get a precision and false positive ratio of 1. We will also be using the ROC curve, with true positive rate on the y-axis and false positive rate on the x axis. For the ROC curve random guessing will give a straight line from TPR = FPR = 0 to TPR = FPR = 1. For both of these curves we will be calculating the area under the curve (AUC), with 1 being the optimal. To calculate these curves and the area under them we will be using the methods `roc_curve`, `precision_recall_curve` and `auc` from the `metrics` module in scikit-learn [5]. We will also be calculating the three measures mentioned with a threshold value of 0.5. For this we will also be calculating the F-score, which is given by $2 \times (precision \times recall)/(precision + recall)$. This is also a measure where a score of 1 implies perfect classification.

To properly compare the performance of the different models, we also optimize each model with respect to a set of hyperparameters. For the xgboost method in particular, there are a large amount of parameters to be tuned, so to further investigate the effects of these parameters, we do the parameter optimization for different fractions of training data, ranging from 0.05% to 80%. For each fraction, we also resample the training data 25 times to obtain results that more well suits the expected values.

To find the optimal parameters, we do not do a grid-based search as this would take too much time, but instead hold all but one parameter constant at a time, looping over all parameters and repeating 5 times, further explained in our earlier report [3].

For the optimization used for the models from which we plot ROC and PR curves, we optimize the models with respect to the k-fold cross validation error with k = 3,4,5 instead of resampling, as this is what we use to evaluate their performance and create the curves.

III. RESULTS

Preprocessing

We first plotted a correlation plot between the 8 predictors and the target class in the data, with the results shown in figure 1.

We see from this that features 6 and 7 along with 2 and 3 correlate strongly with each other, while features 0, 2 and 3 correlate most with the target class among the features. These are the mean, excess kurtosis and skewness of the integrated pulse profile. The weakest correlation with the target class is feature 7, which corresponds to the skewness of the DM-SNR curve. As for the sign of correlation with the target class, we see that the mean and standard deviation of the integrated pulse profile, along with the kurtosis and skewness of the DM-SNR curve correllate negatively, while the mean and standard deviation of the DM-SNR cruve along with the kurtosis and skewness of the integrated pulse profile correlate positively.

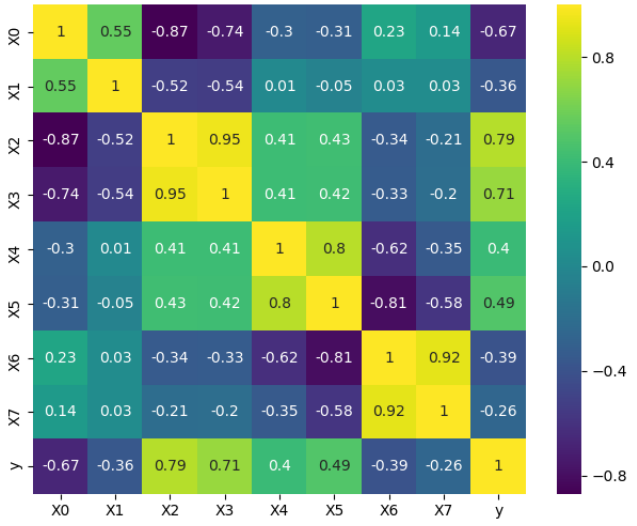

Figure 1. Correlation plot of the 8 features here named X0-X7 in the data, along with the target y. X0-X3 belong to the integrated pulse profile, while X4-X7 belong to the DM-SNR curve. The 4 properties in each are respectively the mean, standard deviation, excess kurtosis, and skewness of the data.

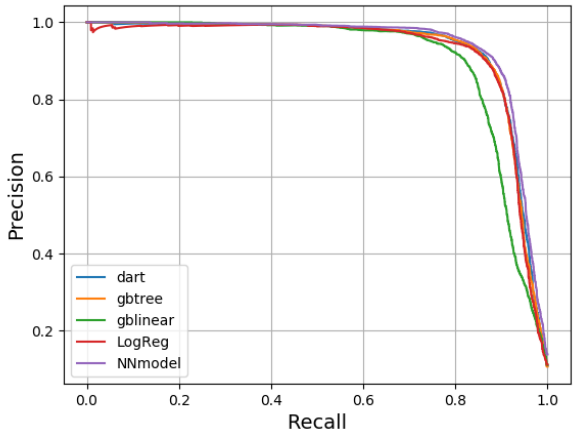

Figure 2. The Precision-Recall curve for the models

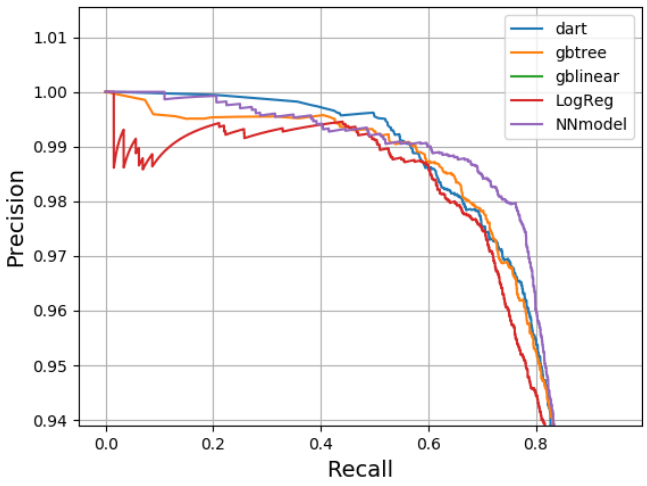

Figure 3. Zoomed in view of the Precision-Recall curve for the different models.

Fitting

When optimizing the hyperparameters of the NN, we had to use a relatively small search space, as the computational time was far greater than for the xgboost derived methods. Excplicitly, we compared one hidden layer of sizes 128,64,32,16 and 8. Epochs ranging from 5 to 20, and batch-sizes of 32, 64, 128 and 256. For the optimized model, we ended up using a hidden layer of 64 nodes, 20 epochs of training, and a batch size of 64.

The logistic regression method had no hyperparameters to optimize, while the xgboost models converged in low enough computational time, that we did not have to predetermine the optimal hyperparameters. The search space for the DART model can be found in Table II, which is also the search space for standard tree booster, without the parameter dr for the drop out rate specific to the DART model.

When plotting the Precision-Recall curve to compare our optimized models, we got the results in Figure 2. A zoomed in view in Figure 3 further shows that the dart model has the highest values of precision for lower values of recall, while the neural netowrk model has the hihghest values of precision for values of recall surpassing approximately 0.6. In the ROC curve plotted in Figure 4, we also see that the curve belonging to the neural network is higher. The models are further compared by the area under the curves which can be found in Table I, which also includes the true positive ratio (TPR), precision, false positive ratio (FPR) and F-score for all models with a threshold of 0.5. In this table PR and ROC refers to the area under the PR and ROC curve respectively.

From the fits made in the XGBoost models, one can also extract the importance given to each predictor, which we did for the gradient tree boosting model (gbtree) in Figure 5. The predictors are numbered from 0 to 7, while 0 to 3 are mean, standard deviation, excess kurtosis and skewness for the integrated profile, while 4 to 7 are same quantities for the DM-SNR curve. Predictor 2 is shown to have the most importance. From the same model, we also plotted one of the decision trees in Figure 6 (at the bottom of the article, as the figure is very large). The features are here labeled from f0 to f7, with the same numbering as in the importance plot.

For the DART XGBoost model [2], we also explored the importance of various hyperparameters in the search space in table II by performing parameter optimization on different training fractions, with the results shown in Table III. Each row contains the optimal hyperparame-

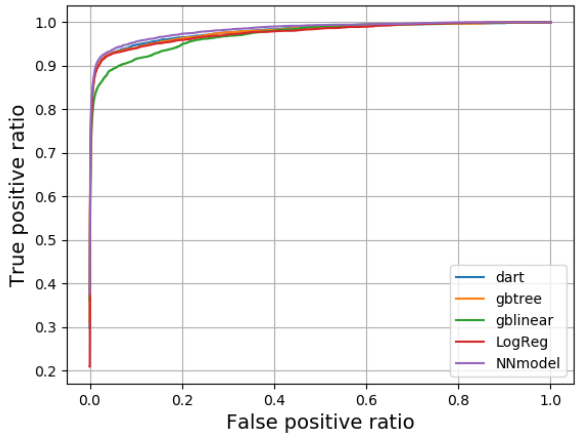

Figure 4. The ROC curve for the models

Model	PR	ROC	TPR	Precision	FPR	F-score
dart	0.931	0.979	0.847	0.928	0.007	0.886
gbtree	0.930	0.979	0.897	0.849	0.016	0.872
gblinear	0.889	0.964	0.584	0.976	0.001	0.731
LogReg	0.924	0.976	0.819	0.941	0.005	0.876
NNmodel	0.939	0.983	0.848	0.935	0.006	0.889

Table I. The area under the precision recall curve (PR), area under the ROC curve (ROC) for the various models. Also includes the TPR (recall), precision, FPR and F-score all calculated with a threshold of 0.5.

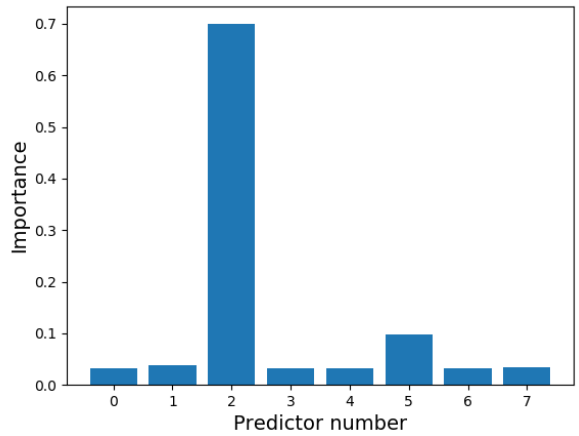

Figure 5. The importance given to each predictor by the tree model in XGBoost. Predictor 0 to 3 are the mean, standard deviation, excess kurtosis and skewness of the integrated profile, while 4 to 7 are the same quantities for the DM-SNR curve

Table II. Search space for optimal parameter search of the DART model. Labels further explained in Table III

md	η	nr	subs	γ	λ	α	spw	dr
1	0.1	3	0.5	0	1	0	0.1	0.0
2	0.2	5	0.6	0.25	1.25	0.25	0.5	0.01
3	0.3	10	0.7	0.5	1.5	0.5	1	0.02
4	0.4	15	0.8	0.75	1.75	0.75	2	0.05
-	0.5	30	0.9	1.0	2.0	1.0	3	0.1
-	0.6	50	0.95	-	-	-	5	0.2
-	0.7	-	0.975	-	-	-	8	-
-	0.8	-	1.0	-	-	-	-	-

trf	md	η	nr	subs	γ	λ	α	spw	dr	AUCPR
0.0005	1	0.5	10	0.5	0	1	1	8	0	0.658
0.001	1	0.3	10	1	0	1	0	5	0	0.782
0.002	1	0.3	50	0.95	0	2	0.25	1	0.02	0.858
0.005	1	0.5	50	1	0.25	1.75	0	1	0.1	0.870
0.01	1	0.4	10	0.95	0	1.75	0.25	1	0	0.889
0.02	2	0.3	15	0.9	1	1.25	1	1	0	0.898
0.05	2	0.3	10	0.975	0	1.75	0	3	0	0.906
0.1	3	0.3	15	1	0.75	1.75	1	1	0	0.917
0.2	3	0.2	50	1	1	1	0.5	0.5	0.01	0.924
0.5	2	0.4	50	0.975	0.5	1.5	0.75	0.5	0.01	0.928
0.6	2	0.4	50	0.975	1	1	1	0.5	0	0.932
0.7	4	0.5	30	1	0.25	0.8752	0.25	0.1	0	0.931
0.8	3	0.4	50	0.975	0.25	1.75	1	0.5	0.02	0.934

Table III. Table showing the optimal parameters as the vary with different amounts of training data denoted as a fraction trf of the total 16,259 data points, using the remaining data as test data. The data was resampled 25 times for each combination of parameters. The column labels refer in order to the training fraction, the maximum depth of the tree, the learning rate η, the number of iterative rounds nr, the subsample ratio subs, parameters γ, λ, α (see xgboost documentation), balance of negative and positive weights spw (used for imbalanced classes), the drop rate dr which separates the DART model from the gbtree model, and finally the area under the precision-recall curve AUCPR.

ters found for a specific training fraction, and the final column contains the corresponding area under the precision recall curve. This area generally increases with increased training fraction, along with the maximum depth of the tree. Except for two outliers at 0.002 and 0.005 training fractions, the value of the number of rounds of training also increases with increased training data.

IV. DISCUSSION

From the PR curve shown in Figures 2 and the ROC curve in Figure 4 we see that the curve for the neural network model is generally higher than the others. For

the precision recall curves this would indicate that the precision of the model does not fall off as quickly if one wants to increase the recall, while for the ROC curves, this indicates that one can obtain a higher true positive rate without getting too much of a false positive rate. However, as we saw in the zoomed in view in figure 3, dart gave higher precision for recall lower than 0.5, so if the goal is to be as sure as possible that what the classifier deems as a positive is indeed positive, this might be better.

For a more general performance on different combinations of precision and recall, we see from the AUC values in Table I that in both the PR and ROC curve the neural network outpreforms the DART and tree models from XGBoost, which again gives better results than logistic regression. One outlier is the linear model from XGBoost, which preformed significantly worse than even logistic regression, especially in the AUC for the PR curve. In the table, we also see the precision and recall values evvaluated at a threshold of 0.5, where we see that although some models have better results than NN at one value, they then have a lower score in on the other.

In the F-score, which considers both precision and recall, we see that the neural network gets the highest score of 0.889, which again strengthens the conclusion that is is overall the best classifier for this case. However, the DART model follows closely with an F-score of 0.886, which may also indicate that the better performance of the neural network is not worth the greatly increased computation time we observed. The tree model for XGB gives similar results as DART in the area under curves, but seems to weigh Recall (TPR) more heavily than precision at a threshold of 0.5, where dart does the opposite. Since the F-score for the DART model is also somewhat better than the gbtree model, this seems to indicate that the drop out parameter which separates the two sometimes activates with a significance. As dropout reduces the amount of active trees, which can be used to prevent overfitting, this would indicate that this may be something worth considering in our dataset, which is why we investigated this further in the differing training fractions in Table III for the DART model.

For the false positive rate, we see that at a threshold of 0.5, the linear booster for XGB gets the best result at 0.001. It does however have a far lower TPR at 0.584 than the other models, which all end up at 0.8 to 0.9. This could perhaps point to that 0.5 is not an especially good threshold for the linear booster, especially if a high TPR is an important trait of the classifier.

In general from the PR-curve, we see that most models reach a simultaneous precision and recall of at least 0.85. This means that around 85 % of all pulsars get predicted as pulsars, and 85 % of all pulsar predictions are actually pulsars. These numbers could be changed somewhat by changing the threshold value for prediction, which corresponds to a movement along the P-R curve. A higher threshold means that more of the positive results get included, increasing the recall value, but also giving more false positives. Similarly, lowering the threshold will result in more pulsars being correctly classified, increasing the precision value, but also giving more false positives. So the optimal threshold will then depend on the wanted characteristics of the classifier.

From the importance plot in Figure 5 we see that predictor 2, the excess kurtosis of the integrated profile is by far the most important feature in the classification, with a score over seven times as high than the other predictors. We also see this reflected in the plot of one of the trees from the XGB tree model in Figure 6. Here we see that all the three top nodes uses the excess kurtosis of the integrated profile to decide. From the correlation plot in Figure 1 we also see that this is a feature which has a correlation of 0.79 with the target, higher than any other predictor. In the correlation plot, one might therefore also expect that predictor 3 corresponding to the skewness of the integrated profile would be almost equally important, as this correlates second most with 0.71 correlation. However, as we see in Figure 5, this is not the case. A possible reason for this can be found by further investigating the correlation plot, where we see that predictors 2 and 3 have strong correlation amongst themselves. Because of this, by including predictor 2, much of the relevant information in predictor 3 is already there, and it does not have to be included separately, which is further illustrated by the predictor not being present in the tree shown in 6. It thus seems that the tree-based architecture used in XGB manages to compress unnecessary information in correlating predictors rather well. Therefore, we chose not to use methods such as principal component analysis that can accomplish the same reduction of dimension, but perhaps not as well as it does not take the target class into account, which XGB does. For the Neural Network however, the unnecessary features may impose a problem, but as there were only 8, we decided against performing any further pre-processing than scaling the data.

With such few predictors, and a relatively large amount of data, we did not seem to encounter any noticeable overfitting in our model comparison, which as mentioned would result in the similarity between the standard gradient tree boosting and DART models. However, when we used smaller varying sizes of training data in table III, we were able to see such effects more clearly. For the maximum depth of the decision trees, we see that this increases with increased training data. The lowest training fraction of 0.0005 results in an average amount of 2 positive cases in the training data, so it seems reasonable that deep trees would overfit heavily in this case, and result in a poor precision-recall curve.

As for the learning rate η, we realize that this is heavily connected to the number of rounds nr, where a lower learning rate would generally take longer, and thus require more rounds to reach convergence. Reaching con-

vergence on the training data may cause overfitting if the data set is too small, so we would expect the product of η and the number of rounds to increase with increased training data sizes. This seems to be the case except for two outliers at trf=0.002 and trf=0.005. We note that these outliers also have the highest values of the drop out rate of 0.02 and 0.1 respectively, which we earlier argued might be used to avoid overfitting. This could therefore be what allows for the increased number of rounds in these cases.

A final parameter that seems to follow a decreasing pattern with increased data size is spw which controls the balance of weights for the positive and negative cases. As the pulsar dataset is imbalanced with about 10 times as many negative cases as positive cases, this is an especially interesting parameter. The general way to scale the positive values in such datasets is by a factor of 10, which would give a value of 10 for spw. However, as we can see in table II, such high values only happen for low amount of data, while for higher amount of data points the opposite happens. This is most likely related to the parameters being optimalized with respect to the area under the precision-recall curve, which through the precision value is also sensitive to the false positives that might appear if the positive cases are weighted too heavily. In the large data case, this seems to result in the negative cases instead being weighed more heavily. In the small data case it also makes sense that weighing positive cases heavily is more important, as the amount of positive cases may otherwise be insufficient for the model to predict any positives at all, which is important for the recall value.

In total, the values in Table III seem to indicate that although overfitting is a problem with lower amounts of training data, this falls off as we approach the typical training fraction of 0.8 of our data. Still, trees of depth 4 are generally not chosen, and the drop rate seemd to have some impact on the difference between the DART and standard gradient tree boosting methods as discussed earlier.

When comparing our results to those of Lyon et. al [4] we see that our neural network seems to give better results than theirs. When looking at the HTRU 2 dataset, ther neural network (MLP for multi layered perceptron) got an F-score of 0.752, while we got 0.889. This is also a better result than their highest score of 0.862 for their Gaussian-Hellinger Very Fast Decision Tree (GH-VFDT). We also see that while their recall at 0.913 for the neural network is higher than ours at 0.848, their precision is much lower at 0.650 vs 0.935 for our NN. This could indicate that their F1 score could be improved by increasing the threshold, so that the precision (number of positive predictions that are actually positive) should go up. If we assume their NN has a similar PR cure to ours, we see from Figure 2 that one could increase the recall up to around 0.8 before the precision starts to drop rapidly.

V. CONCLUSION

By analyzing the pulsar data with different models, we were able to obtain good precision-recall curves despite of the dataset being imbalanced with about 10 times as many positives as negatives. The size of the dataset seemed to be large enough for no considerable overfitting to occur, and we can therefore conclude that our fitted models could be used to predict pulsars from new data with reasonable accuracy. Wherein one could adjust the threshold depending on whether the recall or precision is more important. In particular, we found that a Feed Forward Neural Network [1] performed slightly better than boosted trees [2], both of which with a higher F-score than previous analysis performed by Lyon et. al [4]. Although the Neural Network performed slightly better than XGBoost, we still found that XGBoost converged significantly faster, perhaps making it a more suitable candidate for even larger datasets.

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

[2] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi:10.1145/2939672.2939785. URL http://doi.acm.org/10.1145/2939672.2939785.

[3] Magnus Sikora Ingstad and Fredrik Leiros Nilsen. Performance analysis of feed forward neural networks. nov 2019. URL https://github.com/Magnus-SI/FYS-STK3155/blob/master/Project1/FYS_STK3155_Project2.pdf.

[4] R. J. Lyon, B. W. Stappers, S. Cooper, J. M. Brooke, and J. D. Knowles. Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society*, 459(1):1104–1123, 04 2016. ISSN 0035-8711. doi: 10.1093/mnras/stw656. URL https://doi.org/10.1093/mnras/stw656.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

VI. ADDITIONAL RESULTS

Can be found in the github repository https://github.com/Magnus-SI/FYS-STK3155 under the subfolder Project3

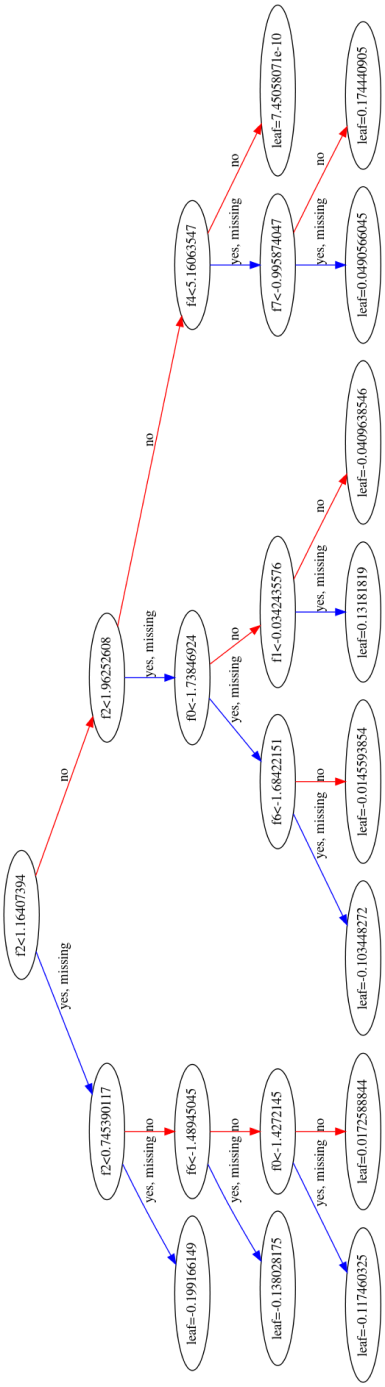

Figure 6. Plot of the tree model from XGBoost. The features are numbered from f0 to f7, with the same numbering as in the importance plot.