# FYS-STK4155 - Project 3

Kristoffer Langstad *
*krilangs@uio.no*

**Abstract**

In this project we use various classification and evaluation methods to analyze the HRTU2 data set containing pulsar candidates. First we will do an analysis of the data set where we make sure that the data set is complete and check correlations. The data set contains 8 continuous features and a binary target class for observing either a non-pulsar or a pulsar. The data set consists of 17 898 instances giving 161 082 data points in total. To analyze this data set we will use various classification models like logistic regression, decision tree, random forest, bagging and boosting to mention some. These models are fitted to the processed data after training and scaling. The we will use various evaluation methods to evaluate errors and scores to see how well the models predict the class targets. The aim is to find the classification model that predicts the pulsar data the best. The results we get are also compared to similar results from a scientific article by Lyon et al. [5] on the same pulsar data set. From theory we would expect the XGBoost classifier to give the best result, since this is one of the most hyped classification algorithms today. From our results we find that the gradient boost classifier give the best overall scores compared to the others with accuracy score of 0.981378. The scores we get for all the classification models are very similar (all higher than 0.977), but the gradient boost gives the highest scores and lowest mean squared error 0.018622. The best classification algorithm in the scientific report, the Gaussian Hellinger Very Fast Decision Tree (GH-VFDT), got an accuracy score of 0.978. This is around the rest of the classification models we have used, but lower than our best.

## 1 Introduction

Machine learning is a more and more used method for classification problems. One application of machine learning is to create filters that decreases the human dependency for classification. Normally a human is needed to go through data and decide what the classification of an object is. With machine learning the goal is to make this as automated as possible by using suited algorithms and methods. We will look at several methods for classification in this project.

In this project we will study a binary classification problem involving the prediction of pulsar star candidates collected during the HTRU survey from the UCI Machine Learning

*https://github.com/krilangs/FYS-STK4155

Repository [8]. This data set, called HTRU2, has previously been studied in a scientific paper by Lyon et al. [5] for finding good filters for classification of pulsars. So some of the results in this paper is used to compare with the results we get. We will use several classification methods from the Scikit-Learn library in Python for classifying the pulsar data, to see which method(s) is the best for classifying these pulsar data. We will explore different approaches from decision trees to random forests, different boosting methods and a couple of other classification methods. For the analysis part we will look at among others the accuracy score, Cohen Kappa score, mean squared error (MSE), confusion matrix and so on.

First we will look at the HRTU2 data set to be evaluated, what it contains and information about the data set. Then in the theory section we will look at the theory of the different classification methods and analysis methods to be used. In the methods section we explain the implementation of the algorithms in Python. Here we first download and read in the data to a Python program. With this program we will look at the data a bit more by looking at the correlation and the information gain of the data features. Then we split the data into training and test data to be used in the various algorithms. For these algorithms we test to find the best input parameters. In the results section we will present and discuss the classification results, and compare them with the scientific report. Lastly, we give a critical evaluation of the methods on which is the best method on the pulsar data set.

# 2 Data

The HRTU2 data set we are analyzing is downloaded from the UCI Machine Learning Repository [8] as a CSV file. The data file contains feature data about possible pulsar candidates collected during the High Time Resolution Universe (HTRU) survey, and are classified as pulsars (positive, 1) or non-pulsars (negative, 0).

Pulsar stars are Neutron stars that rotate with such velocity that they emit radio emission that we can detect on Earth. These stars are often very rare, and are interesting to us in the matter of probes of space-time, interstellar-medium and states of matter [8]. The beam the pulsars emit when rotating are different from pulsar to pulsar, and they may even vary a little for each rotation. So to find a potential signal candidate, the signals are averaged over many rotations of the pulsars. In most cases the detected signals are caused by noise and radio frequency interference (RFI), and not signals from real pulsars.

The data set contains a total of 17 898 number of instances (161 082 data points), where 1 639 are real pulsar examples and 16 259 are spurious examples caused by noise/RFI. So the real pulsar data examples are a minority positive class. All the data have been checked by human annotators. Each candidate is stored in a separate row and contains 8 continuous variables and the class label (0 or 1):

1. Mean of the integrated profile. Renamed: ”Mean_Int”.
2. Standard deviation of the integrated profile. Renamed: ”STD_Int”.
3. Excess kurtosis of the integrated profile. Renamed: ”EX_Kurt_Int”.

4. Skewness of the integrated profile. Renamed: ”Skew_Int”.
5. Mean of the DM-SNR curve. Renamed: ”Mean_DMSNR”.
6. Standard deviation of the DM-SNR curve. Renamed: ”STD_DMSNR”.
7. Excess kurtosis of the DM-SNR curve. Renamed: ”Ex_Kurt_DMSNR”.
8. Skewness of the DM-SNR curve. Renamed: ”Skew_DMSNR”.
9. Target_class. Renamed: ”Target”.

The first four variables are simple statistics from the integrated pulse profile. They describe the longitude-resolved version of the signal that has been averaged in both time and frequency. The next four are obtained from the DM-SNR curve, and contains the same type of information. Their mathematical definitions can be seen in Table 4 in the scientific paper by Lyon et al. [5]. In this scientific paper (section 5) there are also more thorough explanations about the pulsar data and how they are produced.

The 8 continuous feature variables are set as the design matrix $\mathbf{X}$, and the class variable are the target vector $\mathbf{y}$. The data set looks to be complete without any NAN values.

# 3 Theory

First is a theory part about the classification algorithms we use in the program. Then is a theory part about the evaluation methods we use to evaluate the classification algorithms on the data set.

## 3.1 Classification Methods

Under follows a brief theory on how each of the classification methods work. In the Python program, the classification algorithms are implemented using their respective function from the Scikit-Learn library with varying input parameters suited for each method.

### 3.1.1 Logistic Regression

Logistic regression (LR) is used in statistical analysis on a dataset with often several independent variables with a binary outcome. It models the posterior probability of K feature classes by using linear regression to fit data to a logit function yielding a binary output in [0, 1]. When there are more than two outcomes, it is classified by multinomial logistic regression. First is to classify the inputs as class 0 or 1. Here it computes the probability of an observation belonging to the class 1. Then use a logit function for probabilities to classify the elements into the two target classes. Then it defines a boundary threshold for values between 0 and 1, to make sure that all the values are set to 0 or 1. [1]

### 3.1.2 Support Vector Machine

The main idea of support vector machines (SVM) is to find an optimal hyperplane (a line for two feature classes) for K feature classes (making a K-dimensional space) that separates the data points in the target classes. This optimal hyperplane occurs when we have the maximum distance separating the class data points. The support vectors are the data points which are closest to the hyperplane, and has an affect on how the hyperplane looks. The SVM takes the output from the linear function, like the LR method, put now gives it the values [-1, 1]. Then it maximizes the margin between the hyperplane and the data points by using a cost function with a regularization parameter for balancing the loss and the maximization of the margin. Then it minimizes the cost functions with respect to the weights to find the gradients. This is use to update the weights. With no misclassification, the gradients are updated from the regularization parameter. With misclassification, the gradients are updated by adding a loss to the regularization parameter. [2]

### 3.1.3 Decision Tree

Decision trees (CART) uses the data features to construct a tree-like graph for outcomes and possible events. Here each internal node represents a test on a feature that results in leaf nodes representing the class targets. For decision trees the important part is to choose which features to choose as root node and internal nodes, and how to split the features. To decide the importance of the features, we can use the misclassification error, Gini index or the cross-entropy. The Gini index and cross-entropy are better for numerical optimization since they are differentiable. The Gini index shows how good a split is by how mixed the response classes are in the groups created by the split. The cross-entropy is used to calculate the information gain for a specific feature by the difference between the total outcome entropy and the entropy of a specific feature. The feature with the highest value is chosen as the root node. This is then removed to find the next highest feature for each specific branch. This is repeated until the last internal node. The decision tree use a cost function to find the most homogeneous branch when splitting. The stop point of splitting can also be set by us like choosing a certain maximum depth of the tree from the root to a leaf. Pruning is also used to remove nodes that use features with low importance.

### 3.1.4 Bagging

Bagging classification is used to improve CART algorithms by adding bootstrap aggregation and randomization on the classifier. In this project we use bagging on the decision tree classifier. So the bagging classification produces several decision trees to combine several estimates instead of just using the one to try to improve the accuracy score.

### 3.1.5 Random Forest

The random forest algorithm produces several decision trees on bootstrapped training samples which differ from each other with low correlation. It takes a fresh sample of $m \approx \sqrt{p}$ predictors/variables at each split in the tree, with $p$ the full set of predictors. From these splits the best is picked to be split further into sub-nodes. This continues until

a minimum node size is reached. Then it outputs an ensemble of trees to be used for evaluation.

### 3.1.6 Voting Classifier

The voting classifier uses several other classification methods to create an ensemble to try to increase the accuracy score. Since it uses several classification methods it can combine exploit different traits from the used methods. It has two types of voting; hard and soft voting. Hard voting uses the class target with the highest number of votes from the classifiers to be chosen. Soft voting uses the probability vector for each predicted class and are summed up and averaged for each classifier. The class with the highest value is then chosen. [6]

### 3.1.7 AdaBoost

The AdaBoost uses adaptive boosting to sequentially apply weak classifiers to repeatedly modified data sets by combining them into a single better classifier through a weighted majority vote. In this project we use the AdaBoost on the decision tree classifier. The algorithm puts more weight on instances that are hard to classify and less on the instances that are handled well. The algorithm first initializes the observation weights. Then the classifier is fit to the training data using the weights. Then it computes the weighted error rate and another weight given to the classifier to produce the final output classifier. This new classifier is then weighted on the previous classifier, such that the new tree hopefully has a better prediction. The new model is then the two trees added together. This procedure is repeated several times of a given number of iterations. [3]

### 3.1.8 Gradient Boosting

The Gradient boosting is another method for converting weak classifiers into stronger classifiers. The Gradient boost is similar to the AdaBoost, where the biggest difference is how the Gradient identifies the shortcomings of the weak classifiers. The AdaBoost uses high weight data points, while Gradient boosting do the same with gradients in the loss function. This method allows for a more open and specified cost function for optimization that fits the classification problem we are dealing with. [3]

### 3.1.9 Extreme Gradient Boosting

The Extreme Gradient Boosting (XGBoost) is an optimized distributed gradient boosting library made to be highly efficient, flexible and portable. It provides a fast and accurate parallel tree boosting [4]. The XGBoost takes into consideration the potential loss for the possible splits to make a new branch by looking at distributions of the features for all data points in a leaf. This is used to decrease the space of possible feature splits search. This method involves many hyperparameters for optimal tuning.

## 3.2 Evaluation Methods

Since we have a binary classification problem (dichotomous target variable), we can compute the point-biserial correlation coefficient $r_{pb}$ between the data features (sect. 5.1.3 in Lyon et al. [5]). This is a measurement of the linear correlation between the continuous features and the binary target, which yields values between -1 and 1. This is calculated as

$$r_{pb} = \frac{\overline{x}_1 - \overline{x}_2}{\sigma} \cdot \sqrt{\frac{n_2 \cdot n_1}{n(n-1)}}, \tag{1}$$

where $n$ is the total number of samples, $\overline{x}_i$ is the mean value of a feature group $i$ and $\sigma$ is the sample standard deviation. This is calculated with the *pointbiserialr* function from the Scipy.stats library in Python on the design matrix $\mathbf{X}$ and target $\mathbf{y}$. This is calculated for each of the 8 features.

For more information on the correlation in the data set, we can look at the entropy and information gain. The entropy is calculated as ([5])

$$H(x_i) = -\sum_{j \in x_i} P(j) \log_2 P(j),$$

where $j$ corresponds to each value that a feature group $x_i$ can take and $P(j)$ is the probability of $j$ occurring. The conditional entropy of the feature $x_i$ given target $\mathbf{y}$ is

$$H(x_i|\mathbf{y}) = -\sum_{y \in \mathbf{y}} P(y) \sum_{j \in x_i} P(j|y) \log_2 P(j|y).$$

With these definitions we define the information gain (mutual information) for a given feature as the difference between the total outcome entropy and the entropy of the given feature as

$$I(x_i : \mathbf{y}) = H(x_i) - H(x_i|\mathbf{y}). \tag{2}$$

The information gain is a measure of the correlation between a feature and the target which detects non-linearities or the amount of information that one feature provides about another. This is calculated using the *mutual_info_classif* function from the Scikit-Learn library.

For classification we measure the performance of the model with the accuracy score which is the number of correct predictions divided by the total number of predictions:

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n} \tag{3}$$

Here $n$ is the total number of predictions, $t_i$ is the predicted target, $y_i$ is the class target and $I$ is an indicator function as

$$I = \begin{cases} 1, \text{ if } t_i = y_i \\ 0, \text{ if } t_i \neq y_i \end{cases}.$$

The optimal accuracy score is 1, which means that the prediction fits the data perfect. With the *cross_val_score* function in the Scikit-Learn library we also calculate the test cross-validation score with 5 folds with the test data for the chosen classifier.

Another score function is the Cohen Kappa score, which takes into account the accuracy that would have happened through random predictions. It measures how well the classifier actually performs, and the best score is 1 as for the accuracy score. The Kappa score is calculated as ([7])

$$\kappa = \frac{\text{Observed Accuracy} - \text{Expected Accuracy}}{1 - \text{Expected Accuracy}} = \frac{p_a - p_e}{1 - p_e}. \tag{4}$$

For evaluating the error in the predicted values we also calculate the mean squared error (MSE) as

$$MSE(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2, \tag{5}$$

where $\tilde{y}_i$ is the predicted value and $y_i$ is the true value of the variables being predicted of the $i$-th sample.

For more error evaluation we also look at the variance of the predicted values, which is a measure on how far the predictions are spread out from their average value. It is also the represented as the square of the standard deviation. The variance is calculated as

$$Var(\tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(\tilde{y}_i - \frac{1}{n}\sum_{i=1}^{n}\tilde{y}_i)^2. \tag{6}$$

Then we have the bias error, which is the difference between the expected value and the true value. The bias squared is calculated as

$$Bias(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \frac{1}{n}\sum_{i=1}^{n}\tilde{y}_i)^2. \tag{7}$$

With the *classification_report* function from the Scikit-Learn library we can compute a classification report. This classification report contains various averages for the two target classes 0 and 1 (Table 10 in Lyon et al. [5]):

1. Precision: Fraction of retrieved instances that are positive:

$$\frac{TP}{TP + FP}$$

2. Recall: Fraction of positive instances that are retrieved:

$$\frac{TP}{TP + FN}$$

3. F1-score: Measure of accuracy that considers both precision and recall:

$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

4. Support: Predictive positive rate=fraction of positively predicted examples:

$$\frac{TP + FP}{N}$$

5. Accuracy: The accuracy score of the F1-score only (in our case).
6. Macro avg: Average of the unweighted mean per label.
7. Weighted avg: Average of the support-weighted mean per label.

Here we use the prediction results to calculate the precision and recall. They are given as:

Positive (P) - Observation is positive.

Negative (N) - Observation is negative.

True Positive (TP) - Observation is positive, and prediction is positive.

True Negative (TN) - Observation is negative, and prediction is negative.

False Positive (FP) - Observation is negative, and prediction is positive.

False Negative (FN) - Observation is positive, and prediction is negative.

These outcomes are also used in the confusion matrix for binary outcomes. The confusion matrix will look like Figure 1. The values in the confusion matrix are normalized such that the total value of the rows equals 1. The optimal values are TN=1, FP=0, FN=0 and TP=1. This means that all the predictions are correct.

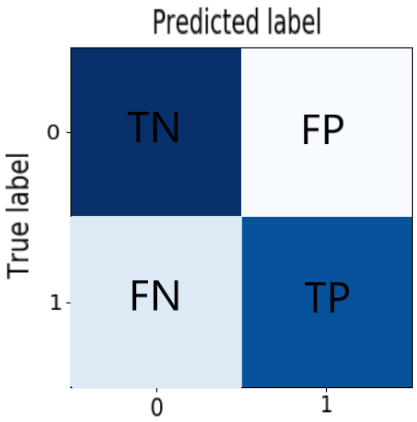


Figure 1: Confusion matrix describing the outcomes of the pulsars. Explanation of the outcomes in 3.2.

Another plot we can compute with these prediction results, is the Receiver Operating Characteristic (ROC) curve. This plots the true positive rate (sensitivity) as a function of the false positive rate (specificity) for the class 0, class 1, micro-average and macro-average for the classifier. The earlier the true positive rate reaches 1.0, the better the accuracy of the classifier.

By plotting the cumulative gain of the class targets as functions of the percentage of samples, we can see how effective the classifier is at predicting the class targets as the percentage of the samples vary.

For the XGBoost we can look at the feature importance when building the trees. This counts the number of times a feature is split across all boosting trees. With the *plot_importance* function in the XGBoost library, we get a bar graph over how many times each feature appear in the trees.

# 4 Methods

The programming is done in Python 3.7 in the program *classif.py* found at the GitHub repository (A), among with various data files and produced figures.

## 4.1 Analyzing the Data

We start by reading in the downloaded CSV file *pulsar_stars.csv* of the pulsar candidates in the program. Then we rename the feature variables to be shorter (as seen in the Data section 2). By using the panda-library, we do an analysis of the data set to make sure that the data file is complete and read in correct. We also check the correlation between the data features by plotting a correlation heatmap using the seaborn-library. Next, we create the design matrix $\mathbf{X}$ and target vector $\mathbf{y}$ as mentioned in the Data section (2). As in the scientific report [5], we calculate the point-biserial correlation coefficient of the data features with the design matrix and target vector. The design matrix and target vector are then split into training and test data with 27% test data. Then we scale the design training and test data with respect to the training data. With the training data we calculate the information gain of the features.

## 4.2 Classification

Then we implement all the classification methods we explained in the Theory section (3.1) into our program using the Scikit-Learn and xgboost libraries. All classifiers have random_state=42, for reproduction of results. The used classifiers include (input parameters that are not covered, uses the default parameters):

1. Logistic Regression (LR): Where we use the *liblinear*-solver.
2. Support Vector Machine (SVM): Where we use probability=True to calculate probability estimates. Set gamma="auto", which is a kernel coefficient as 1/n_features.
3. Decision Tree: Where we set the maximum depth of the tree to 6, since this give the best results. Here we also plot a decision tree for visualization only.
4. Bagging: Where we use bagging on the decision tree with bootstrap, 500 estimators in the ensemble, maximum 200 samples to train on each base estimator and use all processors to run the jobs in parallel (for speed).
5. Random Forest: With 500 estimators.

6. Voting classifier: Use both hard and soft voting separately on all of the mentioned classifier methods mentioned above together (1-5), and use all processors to run the jobs in parallel.

7. AdaBoost: Use AdaBoost on the decision tree with 200 estimators and the "SAMME.R" real boosting algorithm [9].

8. Gradient boosting: Where we use the exponential loss function to be optimized, 200 estimators and a maximum depth of 9 for the number of nodes in the tree.

9. XGBoost: Where we use the "multi:softprob" learning objective with 2 classes (since binary problem), which does a multiclass classification and outputs a vector containing predicted probabilities of each data point to each class. We also use 250 estimators and set maximum depth of the tree to 5. Here we also plot the visualization of the first decision tree in this algorithm, and plot the feature importance of the trees.

10. Voting classifier: Where we use the soft voting classifier on the classification methods yielding the highest accuracy score. This include decision tree, random forest, AdaBoost, Gradient boost and XGBoost.

For each of the classification methods above, we do the same evaluations of the pulsar data set. The various evaluations to be done are explained in the theory section (3.2). After the analysis of the data set, we fit the classification methods to the training data. Then we use the test data to make a prediction of the class labels. With this prediction we calculate the MSE, variance, bias, accuracy score and the Cohen Kappa score. With the test data we also calculate the cross-validation score with 5 folds, where we look at the highest of the 5 and the mean of the 5 scores. We also make a classification report containing various averages for the classification methods. Then we plot the confusion matrix of the predicted results of the pulsars. Then we plot the ROC curve for the classification methods for the sensitivity as function of the specificity. For the hard voting classification we use the prediction, while for the others we use the probability estimation of the test data to make the ROC curve plot. Lastly we use the probability estimate to plot the cumulative gain for the class targets as functions of the percentage of the samples.

# 5 Results

For comparison of our results, we compare with the results in the scientific report by Lyon et al. [5].

## 5.1 Results of the Data Analysis

In Figure 2 we see a heatmap of the correlations between the features (from -1 to 1) in the pulsar data set. A score of 1 means the features are perfectly correlated, while a score of -1 means a perfect anti-correlation between the features. If the score is 0 then the features are independent on each other. From this map we see a high positive correlation between:

- Excess kurtosis of the integrated profile & Skewness of the integrated profile (0.95).
- Excess kurtosis of the DM-SNR curve & Skewness of the DM-SNR curve (0.92).
- Mean of the DM-SNR curve & Standard deviation of the DM-SNR curve (0.80).

This means that these features are highly correlated to each other. The diagonal is the correlation of the feature with it self, which is obviously 1. We also see a high negative correlation between:

- Mean of the integrated profile & Excess kurtosis of the integrated profile (-0.87).
- Standard deviation of the DM-SNR curve & Excess kurtosis of the DM-SNR curve (-0.81).
- Mean of the integrated profile & Skewness of the integrated profile (-0.74).

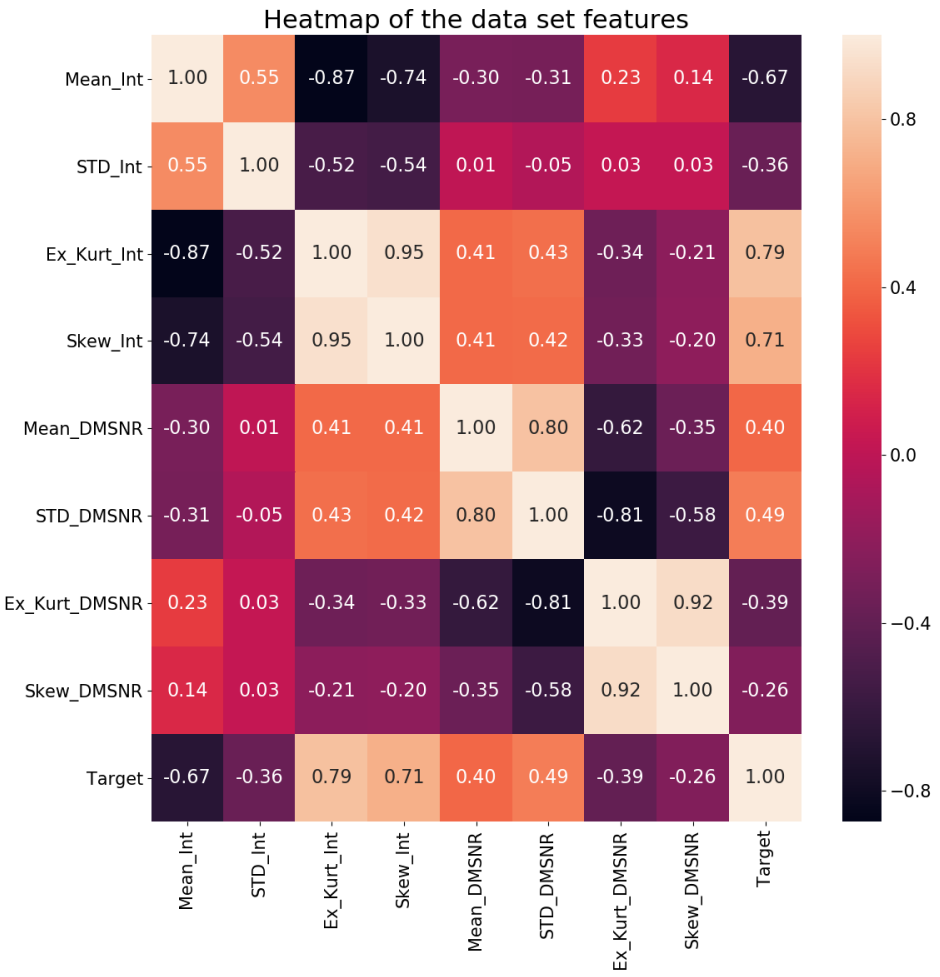


Figure 2: Here we see a heatmap of the correlations between the features in the pulsar data set.

The point-biserial correlation coefficient in equation 1, which measures the linear correlation between the features and the class target, for the features can be seen in Table 1. Here we see that especially the mean, excess kurtosis and skewness of the integrated profile give strong correlations ($> |0.5|$) to the class targets. This is the same values as in Table 6 in the scientific article by Lyon et al. [5] when rounding to the same number of digits, except for the skewness of the DM-SNR curve where we get $\approx -0.259$ while they got $\approx -0.230$. We also see that none of the features are independent on the target class.

After splitting the data set into training and test data and scaling the training data, we calculated the information gain of the features as explained in the Data section 2. The information gains of the features can be also be seen in Table 1 (the rightmost values). There we see that the features of the integrated profile (except the STD) gives the higher values, which means that these features seems to be the most important. The difference between the integrated profile features and the DM-SNR curve features are not that different, so all the feature have to be considered.

| Feature | $r$-correlation | Information gain |
|---|---|---|
| Mean of the integrated profile | -0.673181 | 0.191635 |
| Standard deviation of the integrated profile | -0.363708 | 0.088278 |
| Excess kurtosis of the integrated profile | 0.791591 | 0.226438 |
| Skewness of the integrated profile | 0.709528 | 0.194512 |
| Mean of the DM-SNR curve | 0.4008761 | 0.114824 |
| Standard deviation of the DM-SNR curve | 0.491535 | 0.119248 |
| Excess kurtosis of the DM-SNR curve | -0.390816 | 0.113393 |
| Skewness of the DM-SNR curve | -0.259117 | 0.115064 |

Table 1: Table for the point-biserial correlation coefficient in equation 1 for each feature on the pulsar data set. This is a measurement of the linear correlation between the features and the class target. Specially three features give strong correlations ($> |0.5|$) to the class targets; the mean, excess kurtosis and skewness of the integrated profile. To the right we see the information gain of the features. The information gain is a measure of the correlation between a feature and the target which detects non-linearities. Here we see that the features of the integrated profile (mostly) gives the higher information gain, which means these are the most important features.

## 5.2 Results of the Classification Analysis

After having done all the evaluation methods (sect. 3.2) for all the mentioned classification models (sect. 3.1), we got the printed results in Table 2. In that table we see the values for each of the classification models used for the accuracy score, Cohen Kappa score, highest cross-validation score out of the 5 from the 5 folds used, the mean cross-validation score of those 5, the MSE, variance and bias. The values with bold font are the best values in for that evaluation method. We see that the gradient method gives the best accuracy and Cohen Kappa score and lowest MSE. These are the more important values

for finding the best classification model on the pulsar data. Also the difference in the best values and the corresponding values for the gradient are not that different. All the classification models are very similar in accuracy and error estimate, but from these evaluation methods we can conclude that the gradient boosting classification model seems to be best on the pulsar data we are looking at.

In the scientific report by Lyon et al. [5] they test a few classification algorithms. The best results they get are with the so-called Gaussian Hellinger Very Fast Decision Tree (GH-VFDT). This classification algorithm is explained in the article and can be explained in short as a tree-based algorithm for making the classifier more robust to imbalanced learning problems, like here where the majority of the targets are non-pulsars. This methods only requires mean and standard deviations of the each feature, which minimizes the runtime and memory used. The accuracy score they got with the GH-VFDT algorithm was 0.978, which is slightly worse than what we got for our best gradient boosting classifier. So our classifier seems to be better than in the article, but we have to take into consideration that the data are sampled and split differently which could make a difference in the results.

| Model | Accuracy | $\kappa$-score | Max CV score | Mean CV score | MSE | Variance | Bias |
|---|---|---|---|---|---|---|---|
| LR | 0.977654 | 0.856473 | 0.984504 | 0.977030 | 0.022346 | **0.072618** | 0.083076 |
| SVM | 0.978481 | 0.862088 | **0.986570** | 0.978892 | 0.021519 | 0.072967 | 0.083066 |
| Decision Tree | 0.979723 | 0.874138 | 0.981386 | 0.977859 | 0.020277 | 0.078152 | **0.082955** |
| Bagging | 0.977033 | 0.853283 | 0.985537 | 0.977857 | 0.022967 | 0.073489 | 0.083051 |
| Random Forest | 0.979930 | 0.873158 | 0.983471 | 0.978272 | 0.020070 | 0.075223 | 0.083008 |
| Voting (Hard) | 0.978481 | 0.862979 | 0.985537 | 0.979305 | 0.021519 | 0.074010 | 0.083037 |
| Voting (Soft) | 0.979516 | 0.869426 | 0.985537 | 0.979513 | 0.020484 | 0.073836 | 0.083042 |
| AdaBoost | 0.980550 | 0.878510 | 0.98240 | 0.979308 | 0.019450 | 0.077121 | 0.082971 |
| Gradient Boost | **0.981378** | **0.883433** | 0.985537 | 0.979720 | **0.018622** | 0.076777 | 0.082977 |
| XGBoost | 0.980757 | 0.878907 | 0.983471 | 0.979721 | 0.019243 | 0.075914 | 0.082994 |
| Voting (Best) | 0.980964 | 0.879825 | 0.984504 | **0.980134** | 0.019036 | 0.075396 | 0.083004 |

Table 2: Table for the evaluation values for all the used classifiers on the pulsar data set. From left to right: The accuracy score, the Cohen Kappa score, highest cross-validation score for 5 folds, the mean cross-validation score, MSE, variance and bias. The values in bold font are the best values for that evaluation value. Here we see Gradient boost has the best accuracy, Cohen Kappa score and MSE. The other eval values are not as important as these, and the differences for the other best compared to gradient boost are not that different. From this analysis we can conclude that the gradient boost is the best classifier for our pulsar data set case.

Below we see how the classification report is for the gradient boost classification. We choose to show this one since the gradient boost seems to give the best scores, as we have already seen. For the classification reports for the rest of the classification methods used, see *Class_reports.txt* in the Data folder at the GitHub repository A. From these reports we

can see that the classification models are better at predicting the negative outcome (non-pulsar) than at predicting the positive outcome (pulsar). This is sensible since we have a lot more negative outcomes (in total 16 259) in the data target class than we have positive outcomes (in total 1 639). So it makes sense that the negative predictions are better since the classification models have much more negative outcomes to train on.

```
Classification report for Gradient boost :
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      4392
           1       0.93      0.86      0.89       441

    accuracy                           0.98      4833
   macro avg       0.96      0.93      0.94      4833
weighted avg       0.98      0.98      0.98      4833
```

In Figure 3 we see how the XGBoost algorithm makes its trees by choosing the most important features as root and internal nodes leading to the leafs. This figure shows a part of the first tree the classifier makes. In Figure 4 we see graph bars over the importance features for the XGBoost trees. This figure shows how many times the features appear in the trees. In descending order, we see that the most important features are the excess kurtosis, standard deviation and mean of the integrated profile and standard deviation of the DM-SNR curve when the XGBoost is building the trees.

In the Figures folder at the GitHub repository A we can also see other images of trees made by the decision tree classifier. With these images we see how the trees are made when using different parameters. *tree_ent.png* shows how the tree is made with 6 as the maximum depth of the tree and with entropy and information gain to determine the most important features, while the rest uses the Gini index for best results in our case. *tree_not_train.png* shows how the tree would be made if we did not split the data set into training and test data. *tree_big.png* shows the full tree when not setting a maximum depth of the tree, which we see can make a very large tree. *tree_.png* shows the tree made when we use the optimal parameters for the decision tree classifier. A description of how the trees are made is also well explained in Figure 8 in the scientific report by Lyon et al. [5].

In Figure 5 we see the normalized confusion matrix when using the gradient boost classifier, which gave the best confusion matrix. The sum of the rows equal to 1, and we wanted the optimal confusion matrix with 1 on the diagonal. This scenario says that the classifier predicts the class targets perfectly. This is not the case here. The prediction of a non-pulsar when the observation is a non-pulsar is 0.99, which is very good. The prediction of a pulsar when the observation is a pulsar on the other hand is 0.86, which is not as good. This means that the classifier struggles more to predict a pulsar when the observation is a pulsar. This could come from that the data set contains much more non-pulsar observations than pulsars, as mentioned earlier. The confusion matrices for the other classification methods can be seen in the Figures folder at the GitHub repository A.

From the confusion matrix results where we calculated the true positive rate (sensitivity) and the false positive rate (specificity), we plotted the sensitivity as function of the specificity. These predicted curves are in the case when there is observed a pulsar. This is the ROC curve in Figure 6 with the gradient boost method. The important curves all

go quickly close to 1, where we want the curves to be as close as possible to the top left corner. The rest of the ROC curves plots can be seen in the Figures folder at the GitHub repository A.

In Figure 7 we see the cumulative gain for the gradient boost classifier as function of the percentage of the test data used for the predicted class targets 0 and 1. From the figure we see that the gradient boost is better at predicting the class 1 targets than class 0, since the curve for the class 1 is closer to 1 on the y-axis (further away from the baseline which resembles a random classifier) than the class 0 curve. These results we have already seen in this section in Figure 5 for the confusion matrix for the prediction when we have observed a pulsar. The cumulative gains curves for the other classification models can be seen in the Figures folder at the GitHub repository A.

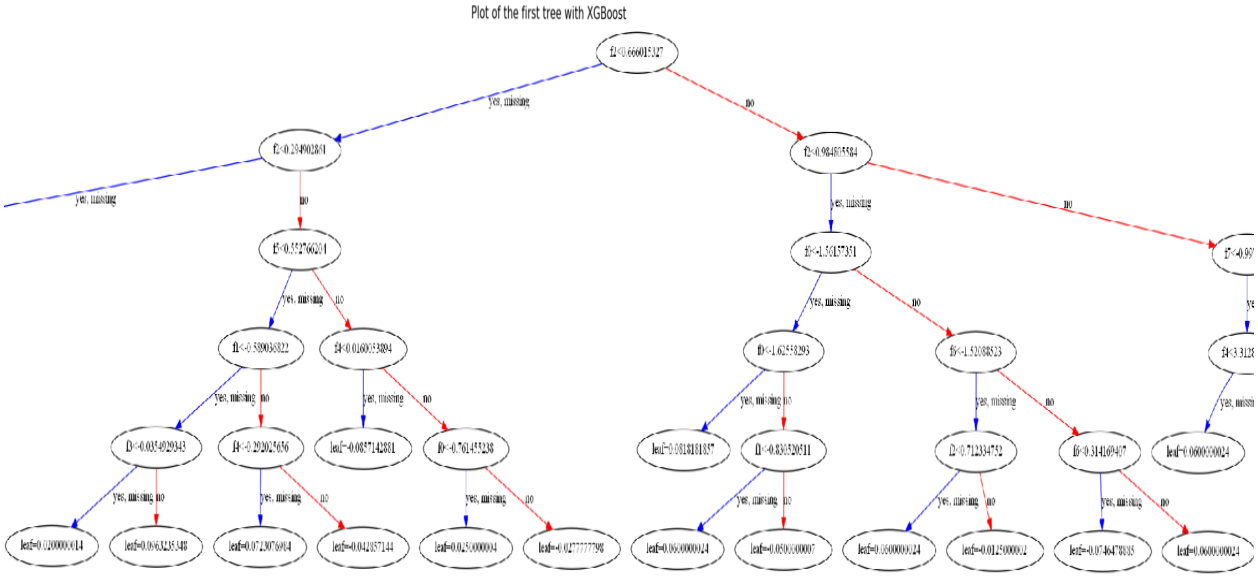


Figure 3: Here we see a part of the first tree just to see how the XGBoost creates the trees (and also other tree-methods). This is just a visualization to show how the trees are made by choosing the most important features as root and internal nodes leading to the leafs. Full image *xgb_tree.png* can be seen in the Figures folder at the GitHub repository A.

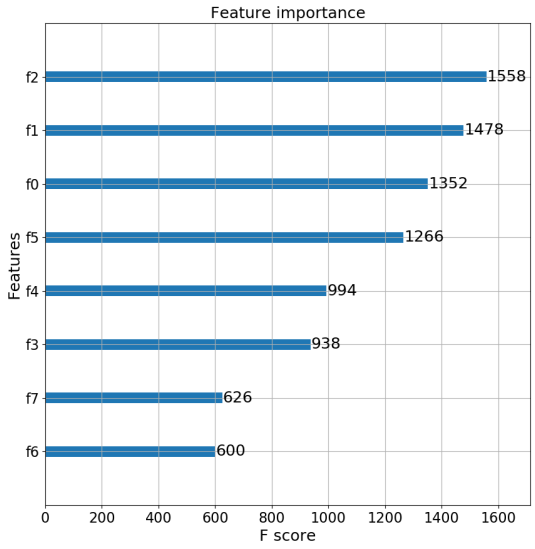


Figure 4: Here we see a plot with bar graphs of the feature importance when building the XGBoost trees. This shows how many times a feature is split across the boosting trees for this model. They are ordered after how many times they appear in the trees. The features are labeled from f0-f7 which represent the data features in the Data section 2 (from 1-8) respectively.

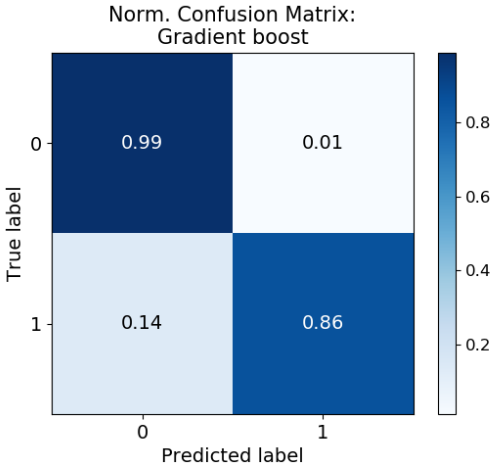


Figure 5: This figure show the normalized confusion matrix made with the gradient boost classifier. This is the classifier that gave the best accuracy score and confusion matrix. See that the classifier is much better at predicting the true negative (TN: 0,0) than the true positive (TP: 1,1). This means that the classifier is better at predicting a non-pulsar which by observation is a non-pulsar, than it is at predicting a pulsar which is observed as a real pulsar.

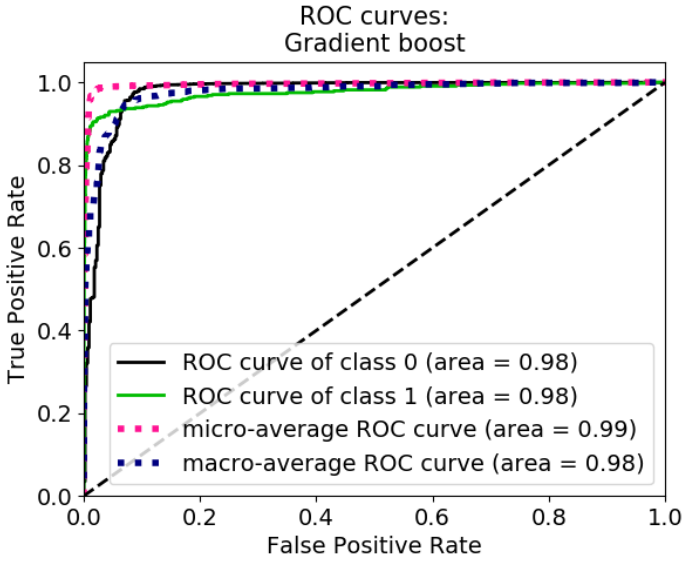


Figure 6: Plot of the Receiver Operating Characteristic (ROC) curve with the gradient boost classifier for plotting the true positive rate (sensitivity) as function of the false positive rate (specificity) for target classes 0 and 1. Here we see that the curves early reach 1, which means good accuracy of the test data.

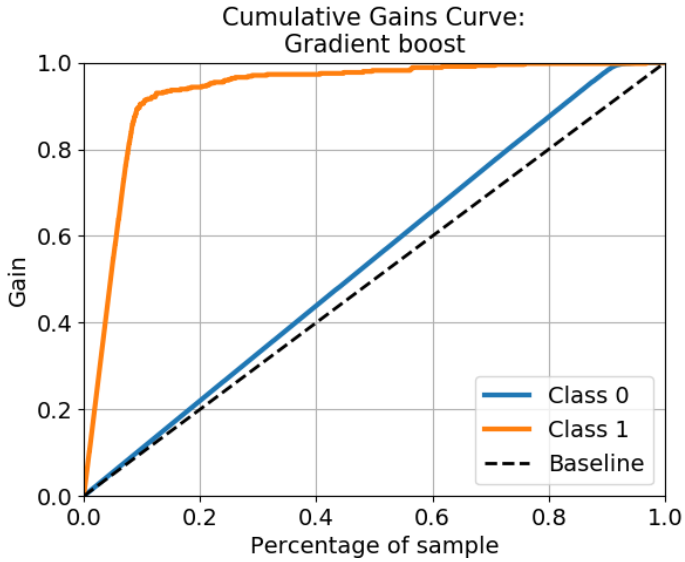


Figure 7: Plot for the cumulative gain for gradient boost classifier as function of the percentage of the data used. We want the curves to go to 1 as early as possible for best results. Here we see that the model does worse when predicting the class 0 than class 1 when we have observed a pulsar, which we already have seen from our results of the confusion matrix.

# 6 Conclusion

In this project we have analyzed the HTRU2 data set containing pulsar candidates by using various classification to see how well they predict the binary target classes. To do this we have used various evaluation methods for evaluating errors and scores. For the classification methods we have tested different input parameters to optimize each classifier. After all the analysis we found that gradient boost classifier gave the best overall scores and errors with accuracy score of 0.981378. The other classifiers all had similar results as the gradient boost, but this gave slightly better overall results. From our plots we also found that the classifiers are better at predicting the observed non-pulsars correct (0.99), than predicting the observed pulsars (0.81-0.86). We also compared our results to results in the scientific article by Lyon et al. [5], which found the best accuracy score of 0.978 with the GH-VFDT classification algorithm on the same data set.

For future work we could try to optimize the classifiers even more by testing the input parameters more. We could also try to change how we sample and train/split the data set for different results. We could also try to implement other classification methods like Neural Networks, the GH-VFDT algorithm from the scientific article or even try to combine methods for improved results. Another thing that could be done is to change and use these methods to build automatic filters to classify pulsars, which earlier had to be classified manually by humans.

# A Appendix

Link to GitHub repository:
https://github.com/krilangs/FYS-STK4155/tree/master/Project3

# References


[1] P Chandrayan. Logistic Regression For Dummies: A Detailed Explanation. *Towards Data Science*, 08 2019. URL https://towardsdatascience.com/logistic-regression-for-dummies-a-detailed-explanation-9597f76edf46.

[2] R Gandhi. Support Vector Machine — Introduction to Machine Learning Algorithms. *Towards Data Science*, 06 2018. URL https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. **The elements of statistical learning**: data mining, inference and prediction. *Springer-Verlag, New York*, 2009. ISSN 0172-7397. doi: 10.1007/b94608. URL https://doi.org/10.1007/978-0-387-84858-7.

[4] Morten Hjorth-Jensen. Data Analysis and Machine Learning: From Decision Trees to Forests and all that. https://compphysics.github.io/MachineLearning/doc/pub/DecisionTrees/html/DecisionTrees.html, 2019.

[5] 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.

[6] S Mangale. Voting Classifier. *Medium*, 05 2019. URL https://medium.com/@sanchitamangale12/voting-classifier-1be10db6d7a5.

[7] M.L. McHugh. Interrater reliability: the kappa statistic. *Biochemia Medica*, 22 (3):276–282, 10 2012. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900052/.

[8] UCI Machine Learning Repository. HTRU2 Data Set. https://archive.ics.uci.edu/ml/datasets/HTRU2. Retrieved: 03-12-2019.

[9] J. Zhu, S. Rosset, H. Zou, and T. Hastie. Multi-class AdaBoost. 01 2006. URL https://web.stanford.edu/~hastie/Papers/samme.pdf.