Benchmarking Models in Classifying Pulsar Stars

Anthony Cavallero
Computer Science and Engineering Department
The Ohio State University
Columbus, Ohio

Nicholas Kashani Motlagh
Computer Science and Engineering Department
The Ohio State University
Columbus, Ohio

Tong Liang
Computer Science and Engineering Department
The Ohio State University
Columbus, Ohio

Rohit Rajendran
Computer Science and Engineering Department
The Ohio State University
Columbus, Ohio

Abstract— **Classifying and finding new pulsar stars are of great interest to the scientific community. Pulsar stars provide a glimpse into the still unknown mechanics of the universe around us in the way they manipulate the fabric of space around them, and how their emissions travels through the universe. Applying machine learning in this context allows for researchers to quickly identify new candidate pulsars from a diverse pool of measurements. This body of research explores the effective performance of different classification models in identifying pulsar stars from a large pool of stellar emissions data from the High Time Resolution Universe Survey (HTRU).**

Keywords—Pulsar, Classifier, HTRU, Machine Learning

I. Introduction

We will first begin this exploration with a familiarization of the term pulsar star. A pulsar star is a specific class of neutron star. This neutron star classification is reserved for a massive star at the end of its lifecycle that has undergone a gravitational collapse of its stellar mass [1]. This gravitational collapse causes protons and electrons to collapse into one super dense collection of neutrons aptly named a neutron Star. Current estimations put forth that a neutron star is anywhere between 1.3 to 2.5 times the mass of our sun, packed into a sphere of roughly 12 miles in diameter.

The specialized subset of neutron stars known as pulsar stars are newly formed neutron stars that, by the Law of Conservation of Angular Momentum, begin to collapse on themselves: their rotational speed increases as they condense. As these stars rotate, they eject radiation out of jets formed from the strong magnetic fields from high speed rotations. These jets fall upon the poles of the magnetic fields and emit waves on the spectrum from radio to gamma rays [2].

We observe these jets sweeping across our sensors (radio telescopes etc.) as pulses over a given period. These pulses occur at regular intervals and can be sampled and integrated together into a single-phase integrated pulse profile that describes the average rotational period of a pulsar star. In addition to the pulse profile, another key feature of the pulse profile is *dispersion measure* (DM). The DM calculates how the waves that traverse space and interstellar medium (gasses, debris, etc.) are dispersed and distorted.

By looking at the statistical measures of mean, standard deviation, skewness (majority of data favors a left or right extreme), and kurtosis (measurement of outliers present in the data.) we can apply different machine learning algorithms to learn underlying characteristics of pulsar stars versus other pulsar like transmissions in the universe. The methodologies described below highlights the teams' approach to different learning algorithms used to classify a measure as pulsar or not pulsar based on the HTRU dataset on the features of mean, standard deviation, skewness, and kurtosis of both the *integrated pulse profile* and *dispersion measure*.

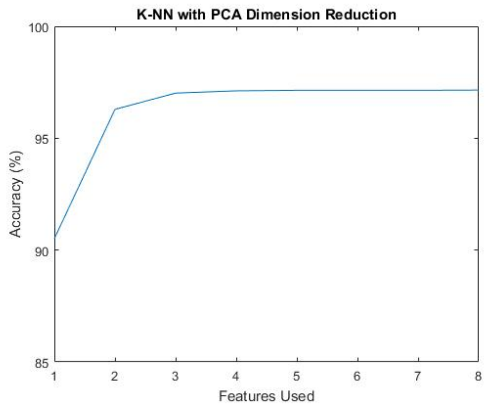
II. Methodologies & Results

A. *Basic Models: K-Nearest Neighbors (K-NN) and Support Vector Machines (SVM) with Principal Component Analysis (PCA)*

The first models covered will be the relatively simple K-Nearest Neighbors (K-NN), and Support Vector Machines (SVM). K-NN models work by finding the point(s) in the training set that is closest, based on Euclidian distance, to a given test point and then classifying that test point to the most frequent class among the selected training points. Our baseline K-NN model (implemented with MATLAB's fitcknn function) with k equal to 5 had averaged accuracy of 97.2% over 5 runs with randomized test and training sets. As a default comparison model, the Simple majority classifier can be used, which would merely classify every point to the most frequent class. In our dataset, there were 1639 data points that were pulsar stars out of a total of 17898 points, so the simple majority classifier would have accuracy of roughly 90.8% by classifying every point as not a pulsar. As such, this baseline K-NN model had an improvement of 6.6% over the default classification without any tuning. The key parameter in K-NN models is k, the number of nearest neighbors to select. A small k value leads to a model with higher susceptibility to outliers whereas a very large value of k will simply degrade the model into a simple majority classifier. In our dataset, the 1-NN model starts with accuracy of roughly 96% and quickly reaches the peak accuracy at 5-NN, which stays approximately the same up until around 40-NN. After the point of k equal to 40, the accuracy starts to slowly fall as k is increased.4

SVM models work instead by creating a division line between the two classes and maximizing the margin, the distance from the closest points on either side to the line. Given that our dataset is not linearly separable, there is no perfect division line which can fully separate the pulsar data points to the non-pulsar data points, so we will have to use a soft margin. In this situation, a parameter c is introduced to adjust the level of slack, which pertains to the prioritization of error reduction. A large value of c will minimize the error in the training set with little priority for margin maximization, whereas a small value of c will maximize the margin at the cost of having possibly increased error. Having a value of c that is too large can result in overfitting the data, as the training error will be very small, but due to the corresponding small margin, the model will be very fine tuned towards the data in the training set. Having a value of c that is too small can also be detrimental in that the model will have little priority on reducing errors and as a result have decreased accuracy. With a c value of 1, our base SVM model (implemented with MATLAB's fitcsvm function) had accuracy of 98%, which is roughly 7% higher than a simple majority classifier and 1% better than the K-NN model. One disadvantage of the SVM algorithm is that it uses dot products for calculation and as a result is not nearly as scalable as K-NN and as a result the time needed to train the SVM model is substantially longer for larger datasets such as the one we are using. (In this instance, the K-NN model took a few seconds to train whereas the SVM model took upwards of 5 minutes to train).

To avoid such hefty training times, data preprocessing techniques such as principal component analysis (PCA) can be used. PCA is a technique to transform some number of correlated variables into a smaller set of uncorrelated variables which are known as principal components and are based on the matrix's eigenvectors [3]. As such, PCA can be used to find the most informative features of a given dataset, which allows for the discarding of the least important features, which often may contribute very little to the classification. Discarding these features then reduces the dimensions of the data matrix to allow for faster computations. An important distinction to be made is that the PCA analysis must be applied to just the training set (not the whole data set) and then this transformation must be projected on to the test set. If PCA analysis is done separately for the training and test set, then there may be a feature mismatch as the most important features from the training set may not match directly with the most important features of the test set. This transformation also cannot be performed on the entire data set, or else information from the test set would be included in the training of the algorithm, which would invalidate it. Using PCA analysis on the K-NN model, it was found that using just the 3 most informative features was roughly (within 0.5% accuracy) the same as using all 8 features. Using just 1 feature, however, decreases accuracy to just 90.5% and using 2 features has accuracy of 96.4%. Below is a graph which illustrates this cutoff in PCA dimension reduction.

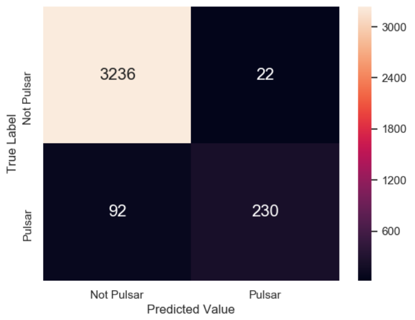


B. *Single Perceptron Baseline with Bagging and Boosting Model Comparisons*

To begin analyzing the Single Perceptron Model the decision of architecture was our first concern. Three possible solutions came about with their own strengths and weaknesses. The first solution that was tossed out rather immediately was implementing the entire learning model as basic python code. This idea was left alone since the overhead of generating the model on top of optimizing the different parameters of the network, which would require much more time than what was initially allotted for our research. Thus, the decision came down to using the open source libraries of TensorFlow or Sci-kit Learn (SkLearn)[4]. While TensorFlow allows for more freedom in model construction and removes much of the complex math of self-implementation, SkLearn provided pre-implemented and optimized model structure that only required a few hyper parameters to be passed to the method to run. Hence the decision was made to use the SkLearn Library.

The Single Layer Perceptron Implemented in SkLearn is a typical implementation of the algorithm. All 8 features of our model are weighted and fed into a single Perceptron whose activates when the weighted sum of the features exceeds the threshold of the Perceptron. This activation in our case leads to a classification of the given features representing a pulsar star.

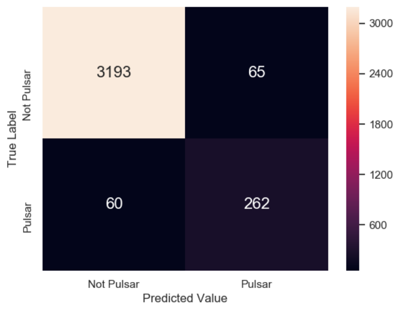
After applying a limited grid search on our hyperparameters of learning rate and maximum iterations, which involved iterating through a fixed step range of parameters, an optimized model with learning rate = .0001, maximum iteration = 100, combined with an L2 loss function and starting with uniform weights for each feature yielded a model that averaged 96.8% accuracy on a randomly selected test set generated from the whole dataset, with the rest used for training.



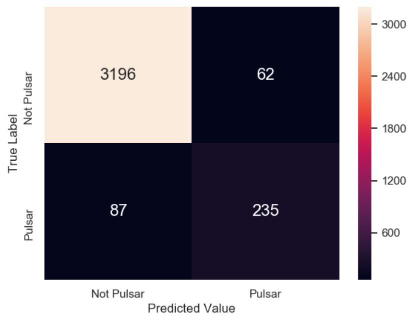
In the figure above, which is a simple confusion matrix of classification of the test set, we see that our model has a higher tendency to misclassify pulsar stars as regular stars. It is good that we have a low number of non-pulsars being classified as pulsars, however we are simply losing to many actual pulsars in this classification. To alleviate this issue, we can compare this baseline model with the ensemble methods of Adaptive Boosting (Adaboost) and simple Bootstrap Aggregation (Bagging). These methods are applied to our weak single perceptron model to try and combat our misclassification issue.

The figure below is the resultant confusion matrix from applying the Adaboost algorithm to our slightly optimized but rather weak single perceptron described above. Both of the Ensemble Methods described above are geared towards improving the models' classification by focusing on the dependence of base models in the Boosting or the independence of base models in the Bagging Case.

In the figure below, which is the confusion matrix for the Adaboost Model we see some increase in correct classification for pulsars. The base model for use in Adaboost was our tuned baseline model. The Adaboost Classifier provided from SkLearn implements Adaboost in the typical fashion, the base model is trained, and then weighed based on how it classified. Models that don't do well are weighted higher so that as we train new classifiers we focus on those harder cases. And we train 50 different estimators in this specific case. In this case the model averaged to an accuracy of 96.4%, which is a slight decrease in accuracy, however what we've sacrificed in accuracy we gained in correctly classifying more pulsars with only a slight increase of non-pulsars being classified as pulsars.

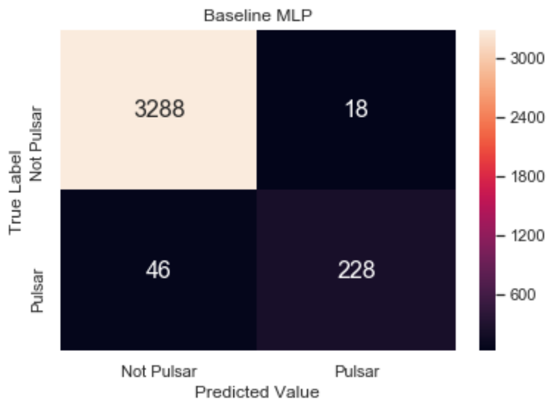


In the figure below, which is the confusion matrix for the Bagged baseline model, we see that this configuration does not offer much improvement over the baseline model and is nowhere near the improvement seen when applying Adaboost despite having an average accuracy of 95.6%. The Bagging classifier provided by SkLearn is implemented by training several of the baseline models on a subset of size 500 of the trainset sampled with replacement and on a limited feature space of 6 features. 15 different estimators where trained and to perform classification a majority function is applied to the output layers of the 15 models. This requires a true majority to vote that a given data point is or is not a pulsar.



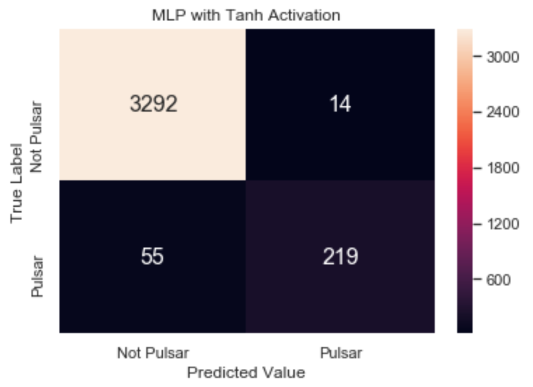
C. *Multilayer Perceptron Baseline with Bagging Model Comparison.*

The Multilayer Perceptron model was taken from the SkLearn Library. This model is a standard implementation of the multilayer perceptron. Training data was normalized with SkLearn's scaler function. From this standard implementation, we trained a baseline model that used one hidden layer of 8 perceptrons. The output of each perceptron was fed into a single output perceptron, which activated if its input was above a certain threshold. The activation function used in the baseline model was *Relu*. This baseline model yielded an average accuracy of 98.11% over randomized test and trained sets.



The figure above records the confusion matrix for the baseline multilayer perceptron model. This model misclassifies pulsar stars more frequently than other stars.

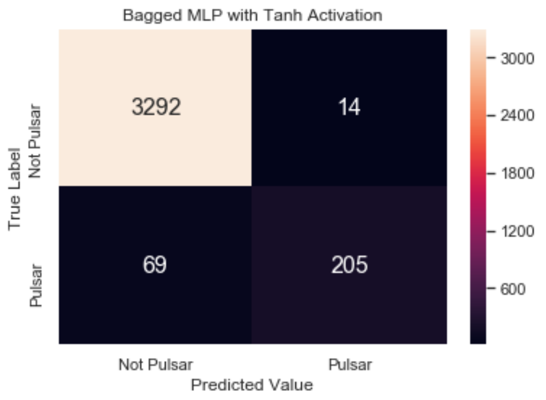
A limited grid search was also applied to the learning rate, maximum iterations, and hidden layer size hyper-parameters. After training models using the *Tanh* activation function and variable hyper parameters, a model with learning rate of .01, maximum iteration of 400, and a hidden layer parameter of (5, 4, 3, 2) starting with uniform weights yielded an average accuracy of 98.54%. Another set of models was generated with the *Relu* activation function. After training models using the *Relu* activation function and variable hyper parameters, a model with learning rate of .001, maximum iteration of 500, and a hidden layer parameter of (4, 4, 3, 2) starting with uniform weights yielded an average accuracy of 98.49%.



The figure above describes the confusion matrix for the tuned multilayer perceptron with a Tanh activation function.

This model misclassified pulsar stars more than the baseline model; however, it classified other stars more accurately.

A simple Bootstrap Aggregation model was created to correct misclassification errors presented in the tuned multilayer perceptron model. The ensemble model used SkLearn's Bagging implementation. The tuned model was used as the ensemble's base estimator. The bagged model had an average accuracy of 97.89%.



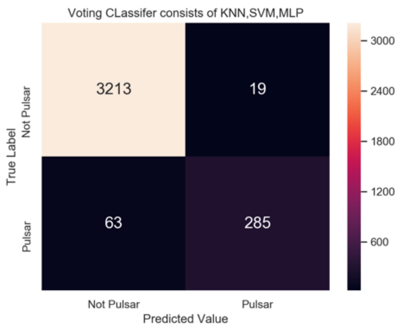
The figure above depicts the confusion matrix for the bagged model. This model misclassified more pulsar stars and an equivalent amount of non-pulsar stars than the tuned model.

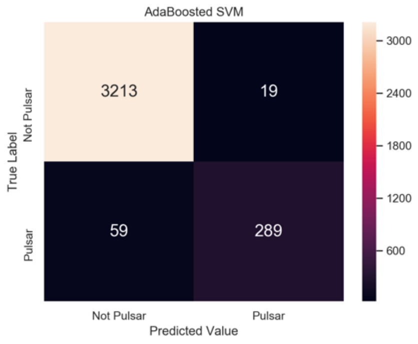
D. *Voting Classifier Baseline with Boosted SVM Model Comparisons*

In our approach, we implemented the Adaboost algorithm with the SkLearn built-in Adaboost classifier equipped with SVM classifiers as our base estimators. The input data is randomly separated into a training set and a testing set. And a data preprocessing transformation is estimated with PCA using the training set for the following training and evaluation.

The baseline model we adopted to compare with our Adaboost ensemble model is acquired by combining 3 baseline classifiers, i.e., nearest neighbor classifier, support vector machine with penalty parameter set to 100 and radial basis function as the kernel, and MLP with one hidden layer, by a voting classifier. All three classifiers are trained with the same training set.

The accuracy acquired by our best Adaboost ensemble model with 3 weak classifiers is 97.82%. The corresponding confusion matrix is shown in following figure. In comparison the accuracy of our baseline voting classifier is 97.71%. The confusion matrices of the 2 ensemble models are shown in the following two figures.





III. Conclusion

The methodologies presented in this report are fundamentally incapable of achieving 100% accuracy due to the linearly inseparable nature of the data. As a result, we can expect the models to accurately classify *most* of the sample points. Through hyper-parameter tuning, we observed that the optimal model for highest accuracy was the multilayer perceptron with 98.54% accuracy: it improved from the simple majority classifier by roughly 7.74% and from the baseline multilayer perceptron by about 0.43%. However, more pulsar stars are misclassified in the tuned model. The negative overfitting of the multilayer perceptron model could be due to the lack of pulsar samples that are in the data set.

The multilayer classifier overfits on non-pulsar stars. In the case that a model attempts to classify an outlier, the sample will be susceptible to misclassification. One change that could improve the performance of these classifiers could be to include more positive samples of pulsar stars. Further research could include implementing and testing Adaboost with the baseline multilayer perceptron as its base classifier. SkLearn does not support multilayer perceptrons as base classifiers with Adaboost ensemble models. With a more balanced test set and multilayer perceptron support for Adaboost, a more accurate model could be constructed.

References

[1] https://www.nasa.gov/mission_pages/GLAST/science/neutron_stars.html

[2] https://www.cv.nrao.edu/course/astr534/Pulsars.html

[3] https://www.stat.cmu.edu/~cshalizi/490/10/pca/pca-handout.pdf

[4] https://scikit-learn.org/stable/index.html