Ensemble methods for PLAsTiCC Astronomical Classification

Youssef Berrada,[*] Georgy Guryev,[†] and Sheng Yao[‡]
Massachusetts Institute of Technology

(Dated: December 12, 2018)

In this paper, we design, test, validate and implement different classification algorithms as part of the Photometric LSST Astronomical Time-Series Classification Challenge. Specifically, we implement a recurrent neural network, a gradient boosting tree and an optimal decision tree algorithm to compare these methodologies and ultimately combine them into an ensemble method to generate better results.

Keywords— *Machine learning, Astronomical classification objects, Gradient Boosting, Recurrent Neural Network, Optimal Tree decision*

The authors gratefully acknowledge Professor Suvrit Sra and Alexander Turner for their valuable advice on the project

All our codes and data are available in the following GitHub repository:

https://github.com/ucfbrd/PLAsTiCC-Astronomical-Classification

[*] yberrada@mit.edu; http://yberrada.mit.edu; https://github.com/ucfbrd
[†] georgy@mit.edu
[‡] shengyao@mit.edu

I. INTRODUCTION

The goal of the project is to develop an ensemble framework that outperform a single implementation of an algorithm such as recurrent neural network or decision based trees. An ensemble method is an algorithm that will combine predictions from outputs of other algorithms in order to make inference. Our goal is to first make a survey of machine learning algorithms for the later defined problem and then pick a selection of three algorithms to elaborate on an ensemble method algorithm. These three models will be optimized separately as standalone algorithms and later combined using a simple neural network of ReLUs.

I.1. Motivation

The application we choose to take on is the The Photometric LSST Astronomical Time-Series Classification Challenge funded by National Science Foundation. The challenge involves classifying astronomical sources that vary in time into different classes, scaling from a small training set to a very large test set of the type the LSST will discover. The data of the astronomical sources is simulated in anticipation of what would actually be observed by LSST. We hope that by experimenting different classification techniques we can provide a superior classifier for labeling the astronomical sources and empower the scientific endeavor of the LSST project.

The project aims to tackle a specific challenge, mainly, how well can we classify objects in the sky that vary in brightness from simulated LSST time-series data, with all its challenges of non-representativity? We hope that by adapting existing classifier to a time-series setting, we can provide a classifier that is tailored to the nature of the task. The classifier will be used to support the actual endeavor of LSST by helping us to categorize the astronomical objects observed, once the training and evaluation are completed on the simulated data.

II. DATA PREPARATION

The dataset we use was collected by numerous members of the astronomical community to provide models of astronomical transients and variables. The dataset includes information on multiple astronomical objects.
The data comes in two forms

- `Metadata`: Summary information about the objects. Each object is uniquely identified by an integer and provides information about the astronomical object such as his localization and intrinsic information.
- `Time-Series data`: The light-curve data come from simulations,which seeks to approximate the actual data which would can be observed from the universe. light-curve data for each object consisting of a time series of fluxes in six filters, including flux uncertainties. The six passbands denoted u,g,r,i,z,y detect light within different wavelength ranges: wavelengths between 300 and 400 nanometers for the u band, between 400 and 600 nm for the g passband, between 500 and 700 nm for the r band, between 650 and 850 for the i band, between 800 and 950 nm for the z band, and be-tween 950 and 1050 nm for the y band. Uncertainties are mainly due to measurement errors or issues related to observations.

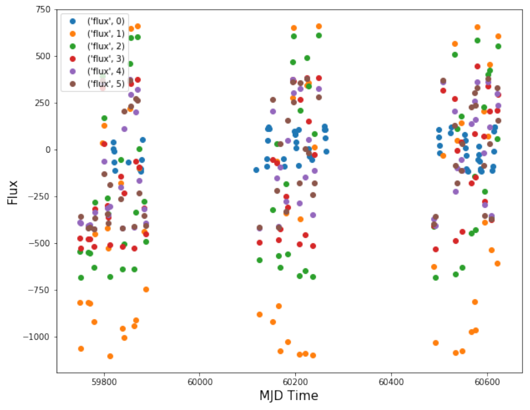

Figure 1. Flux of Object id 615

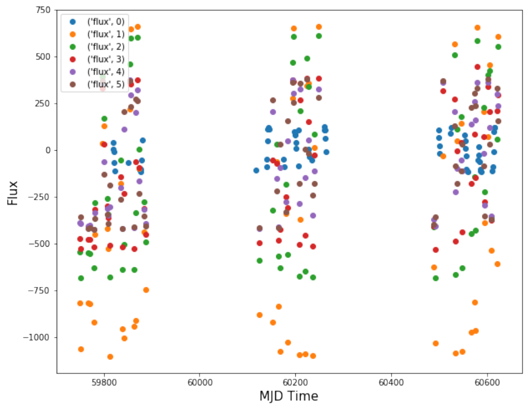

Figure 2. Flux of Object id 130779836

II.1. Data Split

The data provided gives us access to labelled data only for a small portion of the full data, as to what they consider the training set. Hence, the test set is not labelled and out of sample metrics cannot be performed in our setting to report for the class project. We decided to split the training data into a new training data (representing 70% of the original training data set) and a test set representing 30% of the original training data set).

This will give us the opportunity to compare our predictions with the ground truth and analyze our performance on out of sample data. The data split has correctly been done, conserving the class frequency as well as the full data points for each astronomical object. However, after tuning our models on the new defined data set and reporting metrics for the class project and report, we use the full training set to fit our models for submissions on the Kaggle platform.

II.2. Exploratory Analysis

The distribution of classes within the training data is displayed as follows. As we would expect, the types of objects in the universe are not uniformly represented, some object types are more common than others, which gives us some insights in terms of designing the proper loss in classifications i.e. we may want to customize our loss to give better predictions in more frequent classes.

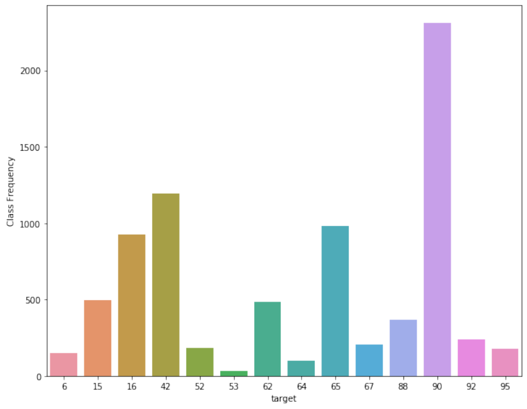

Figure 3. Frequency of Each Class

The frequencies of the passbands are displayed in Figure 4. The passband frequencies seem to be more uniform on a relative basis. As a preliminary analysis, we assume that using data from all passbands should be important as their relative frequencies are within a close range.

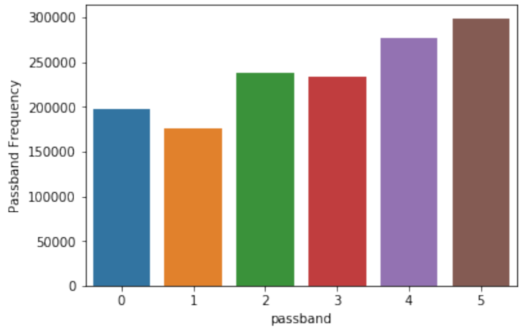

Figure 4. Sample frequency of each passband

The distribution of the flux measurements for each passband is displayed in Figure 5: an important observation from the exploratory analysis is that the measurements of all passbands are susceptible to significant outliers, which again confirm our considerations about experimenting different feature designs in both preserving original data versus aggregation.

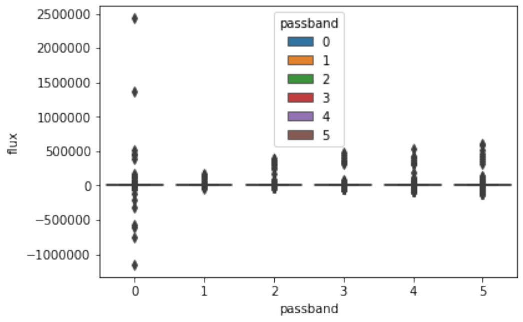

Figure 5. Boxplot of Each Passband

II.3. Feature Engineering

The datasets that we used to predict astronomical bodies are from two types. The first source includes meta-data about the objects such as its location in the sky, red shift, etc. The other source of data includes 6 different light pass-band measurements observed across 2 years (each object is observed during different periods within two years). Our main challenge with feature engineering is to incorporate the time-series data from different frequency band measurements for classification. We experimented with different depth of feature engineering, which we summarize as follow:

- Aggregating the time-series into both static statistics (mean,std and etc) and also time-series related statistics such as auto-correlations and etc.
- Preserve the original series and resort to recurrent neural network to encode the time evolution.

The rationale of experimenting the two ways of working with time-series data is that we think each method has its potential values and disadvantages. By aggregating the time-series, we can get a more robust measure of the characteristics as measurements for each object might be poorly measured or even missing. However, discarding the original time-series means leaving potential valuable information from evolution of the process. On the contrary, feeding the full time-series into models can allow us to preserve the most information, but at the same time the behavior of the model becomes more unpredictable as the time-series data contain both outlying measures and missing entries.

II.4. Time Series Aggregation Features

For the time series features, we adopted `tsfresh` library developed by Maximilian Christ et al. [3]. The library contains an extensive list of feature calculators which can be applied to generated time series features. Specifically, we calculated the following list of features:

- Fourier coefficients of the one-dimensional discrete Fourier Transform for real input by fast Fourier transformation algorithm. We extract the absolute value of the coefficient.

$$A_k = \sum_{m=0}^{n-1} a_m exp(-2\pi i \frac{m \times k}{n})$$

- Mean, Variance, Kurtosis and Skewness of the time series
- Absolute sum of changes over time: calculated as sum of the absolute values of changes in time series.

$$\sum_{t=1\ldots n-1} |x_{t+1} - x_t|$$

- AR(p) coefficients: with p set at 5, the value for p is a simplifying assumption, we did not cross-validate this assumption.
- Linear Trend: coefficient from linear regression where time series is regressed on t.
- Average spectrum magnitude in Fourier domain.
- Sample Entropy: measure of complexity of time series.
- Last Location of Maximum and last location of Minimum.

Although the library provides a full-suite of features, we chose the above as a selective group due to computational reasons. Nonetheless, we believe that the above features summarize the time-series from multiple perspectives.

III. METHODOLOGY

In this section, we will define elements that we used for the loss function, the cross-validation approach and the classification framework.

III.1. Loss Function

The main goal is the accuracy of our predictions and we seek to obtain consistent performance for all categories, which is an important evaluation for the competition. Specifically, we use the following loss function to evaluate our predictions:

$$L = -\frac{\sum_{j=1}^{M} w_j \sum_{i=1}^{N} \frac{1}{N_j} \tau_{i,j} \ln P_{i,j}}{\sum_{j=1}^{M} w_j}$$

where $\tau_{i,j} = 1$ if the i-th object comes from the class j, and N_j is the number of objects in a class j, and w_j are individual weights per class which we decide to equal weight. This designs is aim to give an equal importance to each class we need to classify. Initially we do not have access to the weights, but one way to reverse engineer them is to submit predictions with only one class predicted giving a score and the weights the contribution to the score, we obtain the following score. Therefore we come up with weight approximation based on implicit indicators of class uncertainties. The proposed weights are summarized in the following table:

class	6	15	16	42	52	53	62	64	65	67	88	90	92	95
weights	1	2	1	1	1	1	1	2	1	1	1	1	1	1

Otherwise, one other way could have been to consider the negative log loss function.

III.2. Classification Frameworks

The methodologies that we plan to experiment on for this project are different types or variations of the tree-based methods. We hope to achieve a balance between the accuracy of the models and the interpretability. On one hand, we seek to provide a model with optimal performance, but we also hope to uncover relationships from the data to serve future astronomical discoveries. The list of methodologies that we intend to adopt are the following:

First we would like to apply different decision trees techniques to classify the astronomical objects. For instance, The `CART model` provides intuitive interpretations as it breaks down the decision variable and cutoff at each step, but the 'greedy' nature might not lead to best predictions as the local splits might not be optimal. Then, The `Random Forest` and `XGBoost` are more robust to input perturbations as they add randomness in the training process from variable selection to splitting, but since the final model is aggregated over different 'trees', they are difficult to unravel and are hard to interpret. The `Optimal Classification Tree` is an experimental method that seek to balance between the robustness and interpretability, by re-evaluating an existing decision tree randomly at each node.

In our real-life problem with time series it seems beneficial to exploit the sequential nature of data by using a classifier that would be able to encode and keep track of the temporal information. Based on this intuition we decided to apply the recurrent neural net (`RNN`)to embed temporal information (flux observations) to predict the class of the astronomical object. We will mainly focus on implementing `LSTM` (Long-short Term Memory) RNN.

In order to complete our task we run several classification algorithms. Using the `sklearn` [6] package, we can first run several algorithm to see how they perform. We use a negative log loss metrics as designed above to establish our initial results

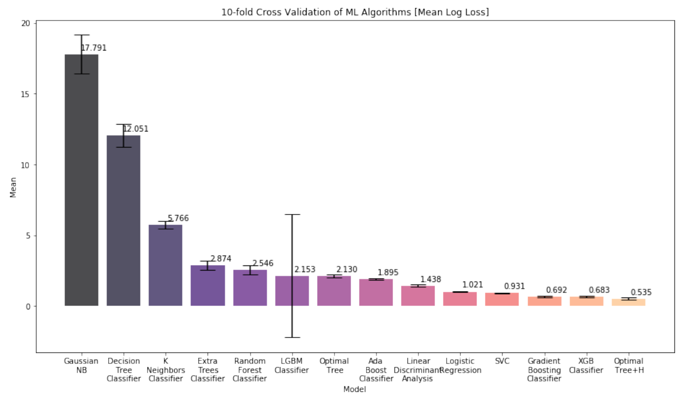

Figure 6. Survey of Machine Learning performance with 10-fold cross validation

After reviewing the initial results from the above first run pass, we decide to focus only on three main algorithm and tune them to optimize the hyper-parameters and then combine the output into an ensemble method that aims to reinforce the classification result. We clearly see that the optimal tree framework offers promising results, especially when implementing it with hyper-planes. We also focus on the `lightGBM` framework as this framework offers gradient boosting algorithm implementation with parallel computing running faster than `xgboost` while providing with similar results in term of accuracy (not on log loss though). We can also plot the run time for the above different algorithms as follows

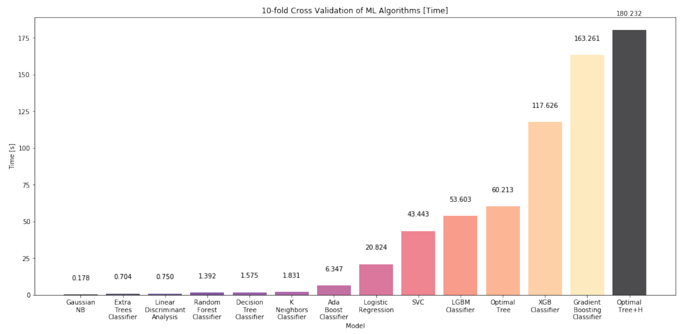

Figure 7. Survey of Machine Learning Run Time with 10-fold cross validation

III.2.1. Recurrent Neural Network

As we briefly mentioned in the previous section in this project we are dealing with the temporal data (flux time series) along with some static meta data. In order to encode the time variations we implement and try a few RNN architectures. The RNN with LSTM cells allows us to encode raw and aggregated time-varying flux information in hidden states. In both cases we use a two-layer LSTM cells (both with 36 or 24 units, tanh/relu activations respectively) with dropout and a dense layer (comprising 14 units) with a softmax activation function.

III.2.2. Gradient Boosting Trees

The next classifier is `gradient boosting`, it recursively combines forecasts from many over-simplified trees. The main idea behind gradient boosting is to construct a stable and powerful "strong learner" by combining "weak learners" in iterative fashion. At each iteration of the algorithm, we construct a shallow tree that minimizes the residual between current and predicted values. Thus, one way to look at the `gradient boosting` as the greedy realization of the empirical risk minimization, where we achieve the objective by minimizing the risk on a space of base learner functions:

$$F_m(x) = F_{m-1}(x) + argmin \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + h_m(x_i)\right)$$

Where $h_m \in \mathcal{H}$ is a base learner function, in GBT $\mathcal{H}$ is a space of shallow trees.

III.2.3. Optimal Classification Trees

Most state of the art decision trees (e.g. CART [2]) exploit the top-down approach to determining the partitions. This approach leads to a number of fundamental limitations: firstly, each tree split is determined in isolation without considering the possible impact of future splits in the tree; secondly such decision algorithms require pruning to achieve trees that generalize well. These both limitations imply a greedy sub-optimal nature. Recently Prof. Bertsimas and Jack Dunn [1] have developed a method that is globally optimal and tractable using mixed integer optimization. The key idea behind the optimal tree is to pose the optimal split/tree growth as the mixed-integer optimization problem that allows to achieve the optimal classifier subject to a class of constant/linear split decisions. Moreover, unlike in standard CART decision trees it is easier to perform multivariate splits than univariate ones and allows to construct truly optimal splits for a fixed maximum depth and minimal number of samples per leaf. Moreover, unlike gradient boosting trees it allows a highly interpretive splits, what turns out to be a valuable property in many areas (e.g. in medical predictions).

IV. RESULTS

In this section, we will outline results from the three selected classification frameworks. We optimize hyper-parameters by selecting the optimal value resulting from a 5-K folds cross validation. We will provide with a confusion matrix on an unseen test set and then discuss.

IV.1. Recurrent Neural Network

To implement the aforementioned RNN encoding with dense layers we used Keras [**?**] and Pytorch [9]libraries. The best performance for our problem is achieved for the two-layer LSTM cells with ReLU activation functions, 24 hidden states and input sequences of length $T = 6$ (i.e. one recurrence per a passband), three dense layers with dropouts and ReLU activation. Thus, the RNN layers encode the time series data, while the dense layers combine the encoded data with static meta data to make an informative decision. From the confusion matrix we see that while RNN allows to achieve a reasonable classification accuracy the problem is complicated by a fair representation of some object classes (primarily 53, 6, 52 and 95).

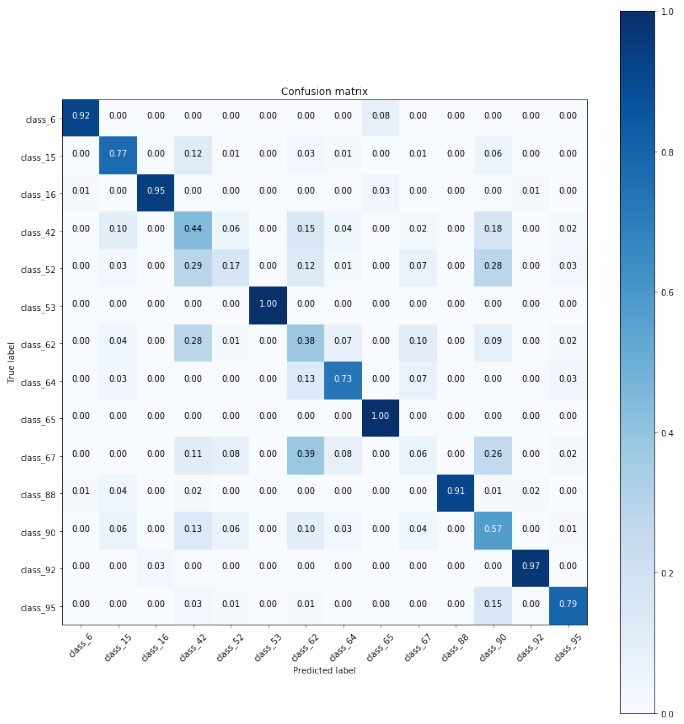

Figure 8. Confusion Matrix for RNN architecture

Metrics	Score
RNN Accuracy	0.6702
RNN Precision	0.6430
RNN Recall	0.6906
RNN F-1 Score	0.6560

IV.2. Gradient Boosting Trees

We implement as detailed in the previous section a light GBM framework [8]. We choose to implement the light GBM framework as opposed to the `xgboost` mainly for tractable algorithm as they should provide similar results. Based on the confusion matrix built from test set generalization, the classifier produces rather accurate predictions for multiple common classes, signaled by the dark blue region on the diagonal axis. As we would expect, the two most frequent classes, 90 and 42, gravitates the model predictions, especially for class 52 and 67 as the model predict 90 or 42 frequently for these two classes. Nonetheless, the errors are not all attributed to the imbalanced data.

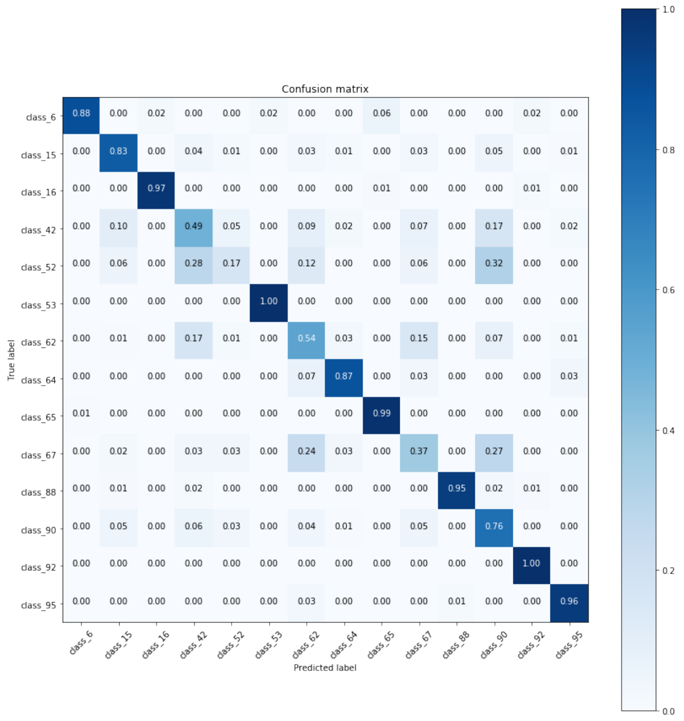

Figure 9. Confusion Matrix for light GBM algorithm

IV.3. Optimal Classification Trees

The performance of the optimal tree model is not ideal compared to the gradient boosting classifier. As shown in figure 7, the classifier suffers from the imbalanced data as many objects from less frequent classes are labeled to be

Metrics	Score
Light GBM Accuracy	0.7633
Light GBM Precision	0.7188
Light GBM Recall	0.768
Light GBM F-1 Score	0.7365

Table I. Performance metrics for the light GBM method

two most frequent class 90 and 42. Possible remedy for that would be to resample the data or adjust the weights for loss for the classes.

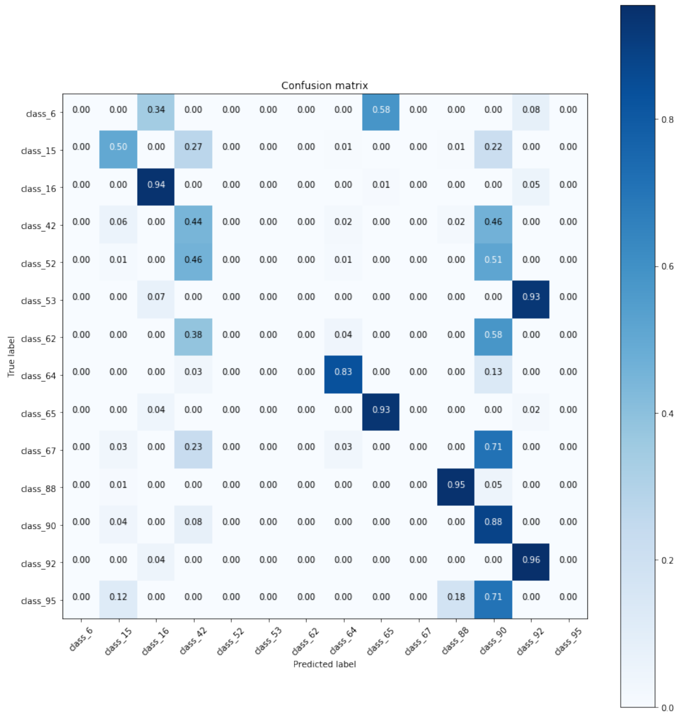

Figure 10. Confusion Matrix for Optimal Classification Tree

Metrics	Score
OPT_H Accuracy	0.67104
OPT_H Precision	0.390091
OPT_H Recall	0.45980
OPT_H F-1 Score	0.418617

V. ENSEMBLE LEARNING

With the different classifiers we built, we believe each of them possess some characteristics/advantages that can complement each other. An example would be that the Recurrent Network takes the entire time series as input whereas the tree models are built on aggregated statistics of time series. The balance between preserving most data and removing noises motivates us to ensemble the predictions from each classifiers. To learn the ensemble weights, we built a simple feed-forward network, whose input layers is simply a concatenation of predicted outputs of each classifiers, we then train this network on the training set of our data. When we validate our model, we will feed the test set predictions from each classifiers into the network and generate the ensemble predictions. The design of the ensemble network is the following:

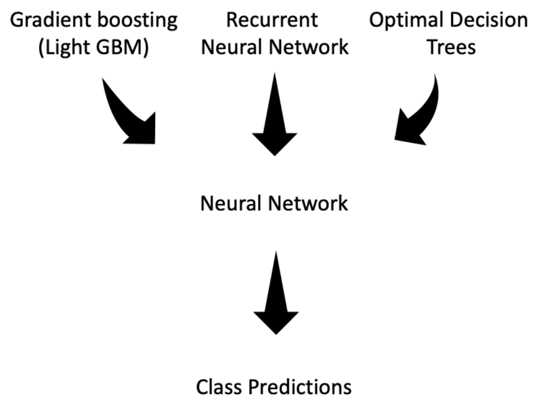

Figure 11. Ensemble Learning Model

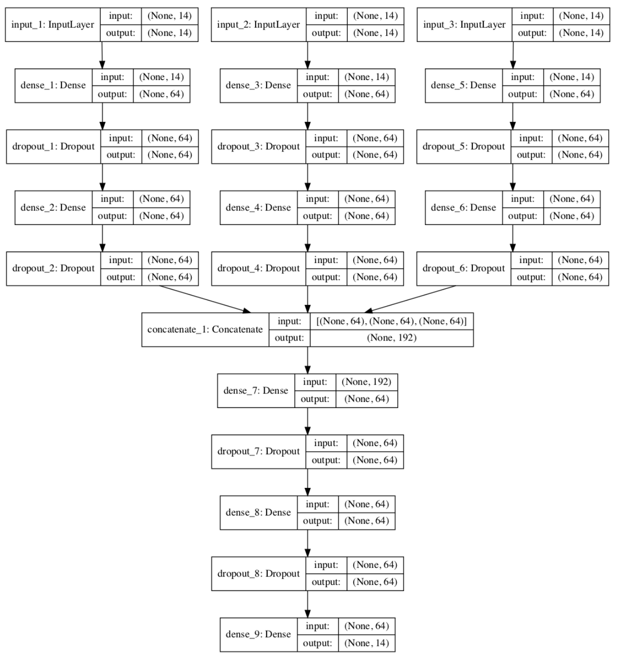

Figure 12. Ensemble Learning Structure

By applying the above, we successfully reach a new optimum in term of accuracy, as we can see in the confusion matrix. We can then calculate performance metrics

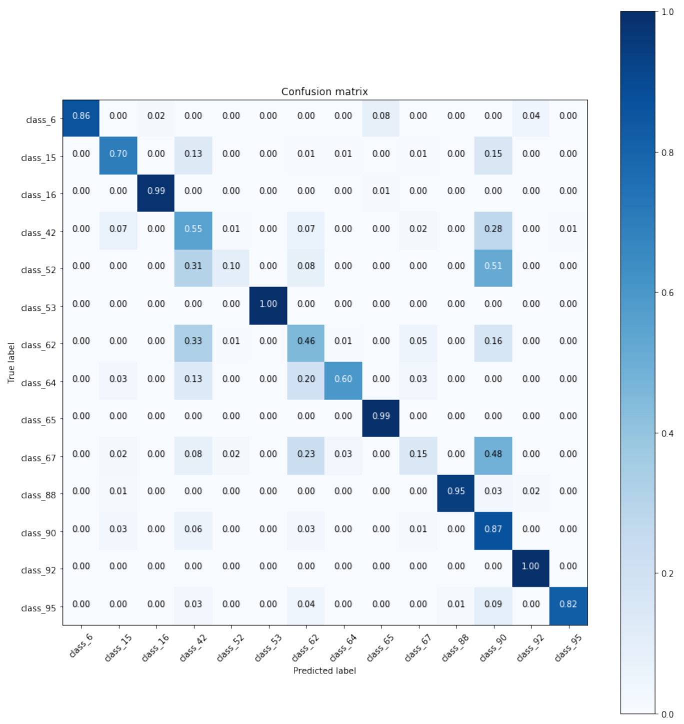

Figure 13. Confusion Matrix for Ensemble Learning Model

Metrics	Score
Ensemble Accuracy	0.7830
Ensemble Precision	0.7758
Ensemble Recall	0.7167
Ensemble F-1 Score	0.7335

Table II. Performance metrics for the ensemble method

VI. DISCUSSION

Over the course of the project, we encountered different challenges at different stage of the process. In the following section, we would discuss what they are and possible remedies that we experimented with and could be extended in the future.

The first challenge comes from the imbalanced nature of the data, which is touched upon when we performed the exploratory analysis and the model results. The extremely unbalanced distribution of classes poses challenges in both training and evaluating our models. First, training on unbalanced distribution would render our model biased towards more frequent classes i.e. predicting common classes tend to reduce the ERM. Applying the model in realized applications would not produce meaningful value above a naive model where we classifies everything into the most common class. To tackle the challenge we considered the following two broadly practiced approaches:

- Resampling: oversample the infrequent classes
- Built cost-sensitive loss functions

In resampling experiment, we oversample the infrequent classes in our training set. The goal is to give more weight to the infrequent classes in training so that the model can learn to distinguish infrequent classes from more frequent ones. However, re-balancing the data distribution would 'sabotage' the model's predicting power in the common classes or even lead to overfitting the infrequent classes. A potentially better design to harness the power of resampling would be to first oversample underrepresented classes in training set and build one-class classifier for each class, then using the same ensemble methodology to train a proper mixing between all one-class classifiers and eventually use for predictions. Due to the limited timeline, we did not implement this design.

Another remedy could be implemented is to assign different penalty for each class in the loss function. By putting more weights on mislabelling infrequent classes, we achieve a more balanced data from the loss-minimization perspective. The challenge with this method is that it introduces another degree of freedom into the process, which entails careful tuning of the weights given to mistakes in each class. By reviewing existing research, it is often reported that cost-sensitive learning outperforms resampling in imbalanced data. ([4] and [5]) An interesting extension for the project would be to compare the resampling versus cost-sensitive methods.

Another challenge we face is the outlier issues with the data. As we demonstrate in the boxplots for each passband, the distribution of the flux measurements are heavily plagued by outliers. Due to the frequent nature of the outliers, we face the challenge of how to better extract useful features from the raw data. One common measure is to build the models after dropping the outliers. Such practice in our project might not be suitable as the outliers might contain valuable information in distinguishing different classes. To tackle the trade-off between signal and noise, we decide to pursue the feature engineering in two different approaches. As we discuss in the feature engineering section, we create two sets of data where we either compute high-level statistics upon the raw data or we simply preserve the raw data with simple rescaling. We believe that the high-level statistics provide us the benefits of removing the noise in the data and also statistics tend to be in comparable scales, which facilities the training of the model. On the other hand, preserving the original data provides us with most complete information, especially the relationships that are discarded in the process of statistical aggregation. Nonetheless, preserving the time series means the model would have to accept the outliers as well, even after we rescale the data.

Our solution to the challenge is to ensemble different models built on each set of feature engineering (Tree models on aggregated data and LSTM RNN on raw time series data), which circumvents the issue of resorting to one specification for prediction. In our project, ensemble model performs better than each standalone models, which can provide some insights into classifier designs in future studies with time-series classification.

VII. CONCLUSION

From the above, we can see that applying an ensemble method outperform single application of algorithms. We have successfully developed, implemented and tuned three classification algorithms before ensemble them into a model that is again optimized by a simple ReLU designed neural network. Our approach helped us benefit from this to achieve better results in terms of accuracy. However, the accuracy might sometimes be of issue and the next step might be to find a tradeoff between an accuracy optimization and a precision, recall or f-1 optimization that can eventually lead us to different results. Another way might also be to include the adaboost or other ensemble methods such as bagging, boosting or stacking.

DIVISION OF LABOR

All three team members contributed equally and significantly to the project.

Appendix A: Bibliography

[1] Dimitris Bertsimas and Jack Dunn *Optimal classification trees.* Machine Learning, pages 144, 2017.

[2] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification And Regression Trees*, Chapman Hall/CRC.

[3] Christ, M., Braun, N., Neuffer, J. and Kempa-Liehr A.W. (2018). *Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh A Python package)* Neurocomputing 307 (2018) 72-77, https://doi.org/10.1016/j.neucom.2018.03.067

[4] P. Domingos. *Metacost: A general method for making classifiers cost-sensitive* In Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 155164, San Diego, CA, 1999. ACM Press.

[5] N. Japkowicz and S. Stephen. *The class imbalance problem: A systematic study* Intelligent Data Analysis, 6(5):203231, 2002.

[6] Pedregosa et al. *Scikit-learn:Machine Learning in Python* JMLR 12, pp. 2825-2830, 2011.

[7] Franois Chollet and others *Keras* JMLR 12, pp. 2825-2830, 2011.,https://keras.io

[8] Guolin Ke et al. *LightGBM: A Highly Efficient Gradient Boosting Decision Tree.* Advances in Neural Information Processing Systems 30 (NIPS 2017), pp. 3149-3157.

[9] Paszke, Adam et al. *Automatic differentiation in PyTorch*, NIPS-W, 2017