Machine Learning for Classification of Astronomical Time Series

May 2, 2019

1 Introduction

The Large Synoptic Survey Telescope (hereafter LSST) being constructed in Chile will discover transient and periodic events in the sky at an unprecedented rate. Once completed, LSST will be 8 meters long and equipped with a 3 billion pixel camera. As a result roughly 20 terabytes of raw data will be obtained during a 24-hour period. Thus far transient and periodic events were classified using spectroscopy. LSST spends significantly less time observing single events than is needed for spectroscopy. Unfortunately there are not enough telescopes in the world combined to keep up with the LSST event rate. As a result the scientific community is turning to machine learning methods to automate this process. To speed along the progress, an open data challenge was held in 2018, the Photometric LSST Astronomical Time-Series Classification Challenge (hereafter PLAsTiCC) [21]. Participants were provided with a training set meant to represent the light sources that have been already classified using spectroscopic methods. They were also given a test dataset with simulated LSST data. The test set was vastly greater than the training set – the first one contained info about 8000 and the second one about 3.5 million objects. This reflects reality, ML pipelines[1] for LSST have to be trained on only the spectroscopic measurements available to us today.

The aims of this essay are to review the relevant scientific and statistical challenges involved, some of the algorithms that have been developed and the metrics by which their performance are evaluated. This is done in 4 parts. In Section 2 I give an overview of the challenge. The scientific motivations behind the choice of the dataset and performance metric are also discussed. Previous machine learning pipelines developed for light source classifications were usually tested on the dataset of the Supernova Photometric Classification Challenge (hereafter SPCC) [13]. A comparison between SPCC and PLAsTiCC is given in Section 3. Furthermore, I give a brief overview of the ML pipelines developed for SPCC. Section 4 looks at the approaches that were taken by the top 5 teams of PLAsTiCC and how useful they would be for LSST. This is done based on the PLAsTiCC discussion board and the slides given by the winners. Finally, in Section 5 I implement three different ML pipelines, compare their performance on the PLAsTiCC dataset and the effect of combining them.

[1]From here on a ML pipeline refers to the combination of a feature extraction method and a machine learning algorithm. Feature extraction is necessary to make raw data less redundant and speed up the training process of the algorithm.

2 Overview of PLAsTiCC

The PLAsTiCC data are separated into a test set and a training set. Both of these consist of a header file which summarizes the astronomical information about the objects and a light-curve file which includes time series of fluxes in six filters including uncertainty of fluxes [21]. The flux of photons coming from an object is equivalent to the absolute brightness of that object at that time. Since the telescope can only use one filter and photograph one part of the sky at a time, the time-series data is very sparse. As is discussed in Section 4, accurately interpolating the missing parts of the time-series greatly improves classification precision.

The difference between the test and training sets is that the header file for the training set also includes the object class. These object classes are represented by random integers so as to not give an advantage to anyone with specialized knowledge about a particular class e.g. type IA supernovae.

Participants of the challenge were asked to submit a matrix of probabilistic classifications. This is in contrast with deterministic classifications where probabilities are reduced to an estimated class label. This reduction comes with a loss in information. For example, probabilistic classification can help determine which cases require a spectroscopic follow-up, a method which is significantly costlier in resource than its photometric counterpart [16]. This however makes it harder to compare the results of PLAsTiCC to those of SPCC since the earlier challenge used a deterministic classifications and thus the performance metrics are going to be incompatible.

PLAsTiCC is hosted on Kaggle, a data science competition platform with a wide userbase. It is likely that the inclusion of participants without specific domain knowledge will result in novel approaches to this classification problem. Indeed, there were over 1000 participating teams, about 100 times as many as the SPCC had. A further advantage of Kaggle is that it provides a discussion board where teams can share their insights and problems thus encouraging collaboration.

A major difficulty for PLAsTiCC participants was that the training dataset did not well represent the test data. In particular, objects in the training set had on average significantly higher fluxes. This is in accordance with reality – spectroscopy is usually carried out on brighter objects. Also, spectroscopy has required better-sampled light-curves than the LSST will be able to offer. As a result, measurements in the time-series in the test dataset be more spaced out.

Another complication is that there is a class of objects that does not occur at all in the training set, class 99. This is meant to encompass objects of scientific interest that have never been observed before but are hypothesized to exist [21].

2.1 Performance metric

Different performance metrics were compared in detail by [16]. It was proposed that the chosen metric has to be suitable for classification without a single scientific goal in mind. A classifier that performs well on all classes should be preferred to one that is tailored for a small subset of the classes. The two proposed metrics were log-loss and Brier score, which are given respectively by

$$Q_n^L = -\sum_{m=1}^{M} \tau_{n,m} \ln \left[p\left(m|d_n\right) \right] \tag{2.1}$$

and

$$Q_n^B = \sum_{m=1}^{M} \left(\tau_{n,m} - p\left(m|d_n\right)\right)^2 . \tag{2.2}$$

Here M is the number of object classes, d_n corresponds to data for the nth object and $\tau_{n,m} = 1$ if the nth object comes from the mth class and 0 otherwise. $p\left(m|d_n\right)$ is the probability of classifying the object as class m given the data. Additionally, weights were added to both of these to deal with the class imbalances in the dataset.

As a sanity check, [16] compares both Brier score and log-loss to the metric chosen for SPCC, the figure of merit (FoM). This was done by looking at the performance of 10 of the ML classifiers proposed in [15]. All three metrics broadly agreed on the ranking of the classifiers, confirming consistency.

Although both metrics showed qualitatively consistent results, weighted log-loss was chosen, owing to its easier interpretability. So to conclude, the loss function for PLAsTiCC is

$$Q_n = -\frac{1}{\sum_m w_m} \sum_{m=1}^{M} w_m \tau_{n,m} \ln\left[p\left(m|d_n\right)\right] . \tag{2.3}$$

It is worth noting that during PLAsTiCC, the used weights were hidden from participants.

3 Previously proposed ML pipelines

A lot of work has been done on transient and periodic classification of supernovas using the dataset from SPCC [13] of 2009. The challenge was more specific in that instead of all light sources, only supernovae were considered. Similarly to PLAsTiCC though, the test dataset consisted of spectroscopically confirmed objects (1,103 objects in total) where as the training dataset was simulated (20,216 objects in total). Note that the total amount of data is vastly smaller than for PLAsTiCC and the difference between training and test sets was not as stark. This explains why potential LSST pipelines needed a new benchmark dataset to be trained on.

The work of [15] discusses possible pipelines for supernova classification, comparing 4 different feature extraction methods and 5 different ML algorithms. The metric they chose to compare different pipelines was the area-under-the-curve (AUC) of the receiver operating characteristic (ROC) curve. The ROC is obtained by looking at the TPR[1] against the FPR[2] as probability threshold is varied.

Three of the feature extraction methods proposed were parametric and model-dependent. Out of those, a template-fitting approach SALT2 [10] that relies heavily on prior knowledge about supernovae performed the best. It can be assumed that this would not generalize well to PLAsTiCC because many other object classes besides supernovae are considered. Fortunately, a fourth non-parametric approach was also included. This was based on first interpolating the missing points in the time-series data using Gaussian processes, then using wavelet decomposition and finally principal component analysis (PCA) to reduce the redundancy in the features.

The machine learning algorithms considered were Naive Bayes (NB), k-nearest Neighbors (KNN), Neural Network (NN), Support Vector Machine (SVM) and Boosted Decision Tree (BDT). It was found that for both the SALT2 and wavelet approaches to feature extraction, BDTs performed best followed by SVMs and NNs. BDT reached an AUC of 0.98 out of 1 which outperforms or is very competitive with other supernova classification algorithms tested on the SPCC dataset. Importantly, it was found that including redshift information did not improve the performance of the BDT which makes it practical for use cases like the LSST.

The SPCC dataset has also been used in developing alert-brokers[3] for the LSST. The Arizona-NOAO Temporal Analysis and Response to Events System (ANTARES), for example, used the data to train a part of their pipeline that deals with supernova classfication [19]. A similar approach to the wavelet/BDT combination of [15] was taken and the impressive AUC score of 0.98 was reproduced.

[1]The true positive rate is given by TPR $= \frac{\text{TP}}{\text{TP+FN}}$ where TP is number of true positives and FN is the number of false negatives.

[2]The false positive rate is given by FPR $= \frac{\text{FP}}{\text{FP+TN}}$ where FP is number of false positives and TN is the number of true negatives.

[3]Alert-brokers are automated systems that sift through, characterize, annotate, and prioritize events for follow-up.

4 Successful approaches to PLAsTiCC

Each of the 5 top performing teams posted an overview of their solutions on Kaggle discussion boards [1], [2], [3], [4], [5]. It is interesting to note that out of those teams only the first place winner, Kyle Boone, has an astrophysical background – he specializes in supernova cosmology. All other teams consisted of veteran Kaggle participators who have applied machine learning to a variety of settings but have no domain knowledge. This becomes apparent in how they approach the challenge.

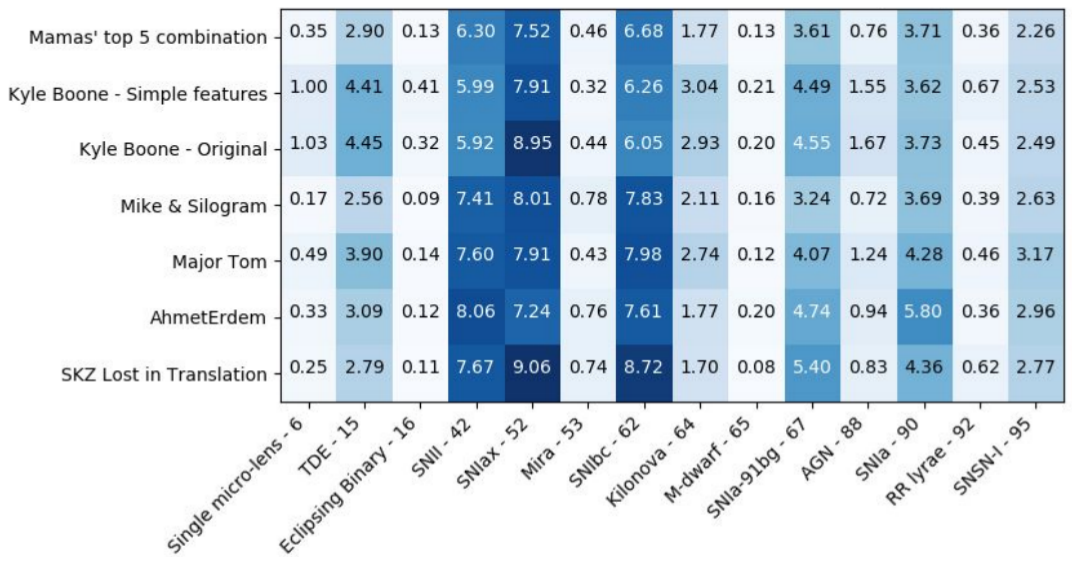
When doing feature extraction all the five teams interpolated the time-series data to fill the gaps. Only Boone used Gaussian processes for this as in the wavelet approach of Section 3. This proved to work better in this case than the Bazin curves and Newling curves used by the other teams. After interpolation various features were calculated for the light-curves such as the maximum brightness, the minimum brightness, the width of the curve, the standard deviation, etc. Another trick that set Boone's feature extraction process apart was that he modified the training set to look more like the test set by degrading the light-curves and modifying redshifts and brightnesses. In the process the size of the training set increased dramatically to include 270,000 objects instead of the previous 8,000.

The ML algorithms used by Kaggle regulars were vastly more complex than Boone's as is demonstrated by Table 4.1. The most popular algorithms was LightGBM, a gradient boosting framework that is often used on Kaggle. Most teams also used a variant of a Neural Network (NN). Both of these are explained in more detail in Section 5. All the regulars used stacking, which is a ML technique where multiple classifiers are combined via a meta-classifier.

Team	**LightGBM**	**NN**	**Stacking**	**Others**
Kyle Boone	yes	no	no	no
Mike & Silogram	yes	yes	yes	no
Major Tom	yes	no	yes	Convolutional Neural Network
Ahmet Erdem	yes	yes	yes	CatBoost
SKZ Lost in Translation	yes	yes	yes	Recurrent Neural Network

Table 4.1: An overview of the ML methods used by the top 5 teams

The overall performances of the teams' pipelines are illustrated by Figure 4.1. It is clear that the biggest contributors to participants' total scores were supernovae. Kyle Boone had the best working pipeline here. This is unsurprising considering that he was able to use his expertise in supernova cosmology during his feature extraction process. Mike & Silogram's pipeline performed better for other classes of objects. This suggests that a pipeline constructed for LSST should be a combination of the two in order to give good classification results over all possible object classes. In fact, user Mamas published the test classifications obtained after combining all the top 5 solutions and ended up with a new best score. This corresponds to the first row of Figure 4.1.



	Single micro-lens - 6	TDE - 15	Eclipsing Binary - 16	SNII - 42	SNIax - 52	Mira - 53	SNIbc - 62	Kilonova - 64	M-dwarf - 65	SNIa-91bg - 67	AGN - 88	SNIa - 90	RR lyrae - 92	SNSN-I - 95
Mamas' top 5 combination	0.35	2.90	0.13	6.30	7.52	0.46	6.68	1.77	0.13	3.61	0.76	3.71	0.36	2.26
Kyle Boone - Simple features	1.00	4.41	0.41	5.99	7.91	0.32	6.26	3.04	0.21	4.49	1.55	3.62	0.67	2.53
Kyle Boone - Original	1.03	4.45	0.32	5.92	8.95	0.44	6.05	2.93	0.20	4.55	1.67	3.73	0.45	2.49
Mike & Silogram	0.17	2.56	0.09	7.41	8.01	0.78	7.83	2.11	0.16	3.24	0.72	3.69	0.39	2.63
Major Tom	0.49	3.90	0.14	7.60	7.91	0.43	7.98	2.74	0.12	4.07	1.24	4.28	0.46	3.17
AhmetErdem	0.33	3.09	0.12	8.06	7.24	0.76	7.61	1.77	0.20	4.74	0.94	5.80	0.36	2.96
SKZ Lost in Translation	0.25	2.79	0.11	7.67	9.06	0.74	8.72	1.70	0.08	5.40	0.83	4.36	0.62	2.77

Figure 4.1: Contributions to the metric from each class (x100) by Kyle Boone.

5 Implementation of ML pipelines

I will look at the performance of three machine learning algorithms: k-nearest Neighbors, a Neural Network and LightGBM. First of these was chosen for illustrative purposes – it is easy to understand and there is only one hyperparameter that can be tweaked. The other two were chosen because they were the most popular amongst top performers. All three are going to be trained on the same dataset i.e. feature extraction will be the same. This makes it easier to compare the performance of the ML techniques.

5.1 Feature extraction

Feature extraction is done differently than the competition winners, namely curve interpolation is going to be skipped. This is done for computational reasons. To illustrate, Kyle Boone ran his Gaussian process method on a machine with 64 Gigabytes of RAM and the code took a day to run. I will be running my code in a Kaggle kernel which offers 16 Gigabytes of RAM. As a result it is appreciated that the implementations will not perform as well as the top teams.

For each object we construct 28 total features using the Python `pandas` library. First, we join the metadata dataset with the light-curve dataset, aggregating the columns of the latter as detailed in Table 5.1. Then some redundant metadata columns are removed, namely `object_id`, `distmod`, `hostgal_specz`, `ra`, `decl`, `gal_l`, `gal_b` and `ddf`. The only metadata columns that are left in our feature set are thus, `hostgal_photoz`, `hostgal_photoz_err` and `mwebv`. The first two correspond to the photometric redshift measurement and its error. `mwebv` corresponds to the extinction of light and is a property of the Milky Way dust along the line of sight to the light source.

Finally, the features are normalized to have mean 0 and variance 1 which is a common practice in machine learning.

Choosing the right set of features in such a competition requires a lot of trial and error unless one is a specialist at astrophysics. To save time, my choice of features was inspired by the discussion boards at Kaggle. In particular, the Kaggle kernel of user olivier proved helpful [6].

Original column	Original column meaning	Aggregating function used
`mjd`	Modified Julian Date	`diff`
`passband`	Which filter is used	`min`, `max`, `mean`, `median`, `std`
`flux`		`min`, `max`, `mean`, `median`, `std`, `skew`, `diff`
`flux_err`	Error of the flux measurement	`min`, `max`, `mean`, `median`, `std`, `skew`
`detected`	Is the object clearly visible	`mean`
`flux_ratio_sq`	Given by $(\frac{\texttt{flux}}{\texttt{flux\_err}})^2$	`sum`, `skew`
`flux_by_flux_ratio_sq`	Given by $\frac{\texttt{flux}}{\texttt{flux\_ratio\_sq}}$	`sum`, `skew`

Table 5.1: Columns of the light-curve dataset and the corresponding aggregating functions used. `diff` is defined as `max - min`.

5.2 ML algorithms

For all of these models m-fold cross validation is carried out to prevent overfitting to the training data. This is done by dividing the training set into k roughly equally sized folds and training a classifier on $k - 1$ of those folds. Then out-of-fold predictions and log-loss are calculated on the remaining fold. This is done m times using a different group of folds every time [11]. As a result we have out-of-fold predictions for the whole training set and we have trained m classifiers. Test predictions are then obtained by averaging the results of the m classifiers. For all our 3 algorithms we have chosen $m = 5$.

To visualize the performance of the ML algorithms, confusion matrices are used. Each row of a confusion matrix gives the average probabilistic classifications of an object class. As a consequence the rows must sum up to 1. These are used instead of, for example, the ROC curves in [15] because they are easier to interpret in the case of a high number of object classes like PLAsTiCC.

5.2.1 k-nearest Neighbors

KNN is a simple algorithm which memorizes the training dataset and then classifies a given object based on a majority vote amongst its k nearest neighbors [8]. The nearest neighbors are found based on Euclidean distance so it is immediately clear how important it was that the features were normalized. The only hyperparameter for this model is the number of neighbors, k. The basic working principle of KNN is illustrated by Figure 5.1.

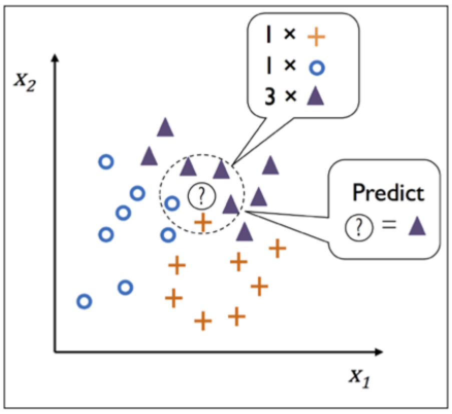


Figure 5.1: KNN demonstration for $k = 5$ and 2 features x_1 and x_2. Total number of object classes is 3. Source: [20]

Here the method is implemented using the `scikit-learn` library. To choose the hyperparameter k the resulting log-loss metrics for k in range 2 to 22 were measured. This is shown in Figure 5.2.

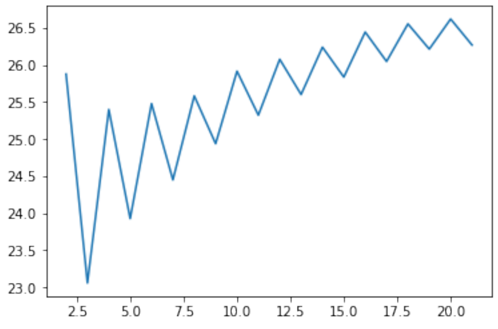
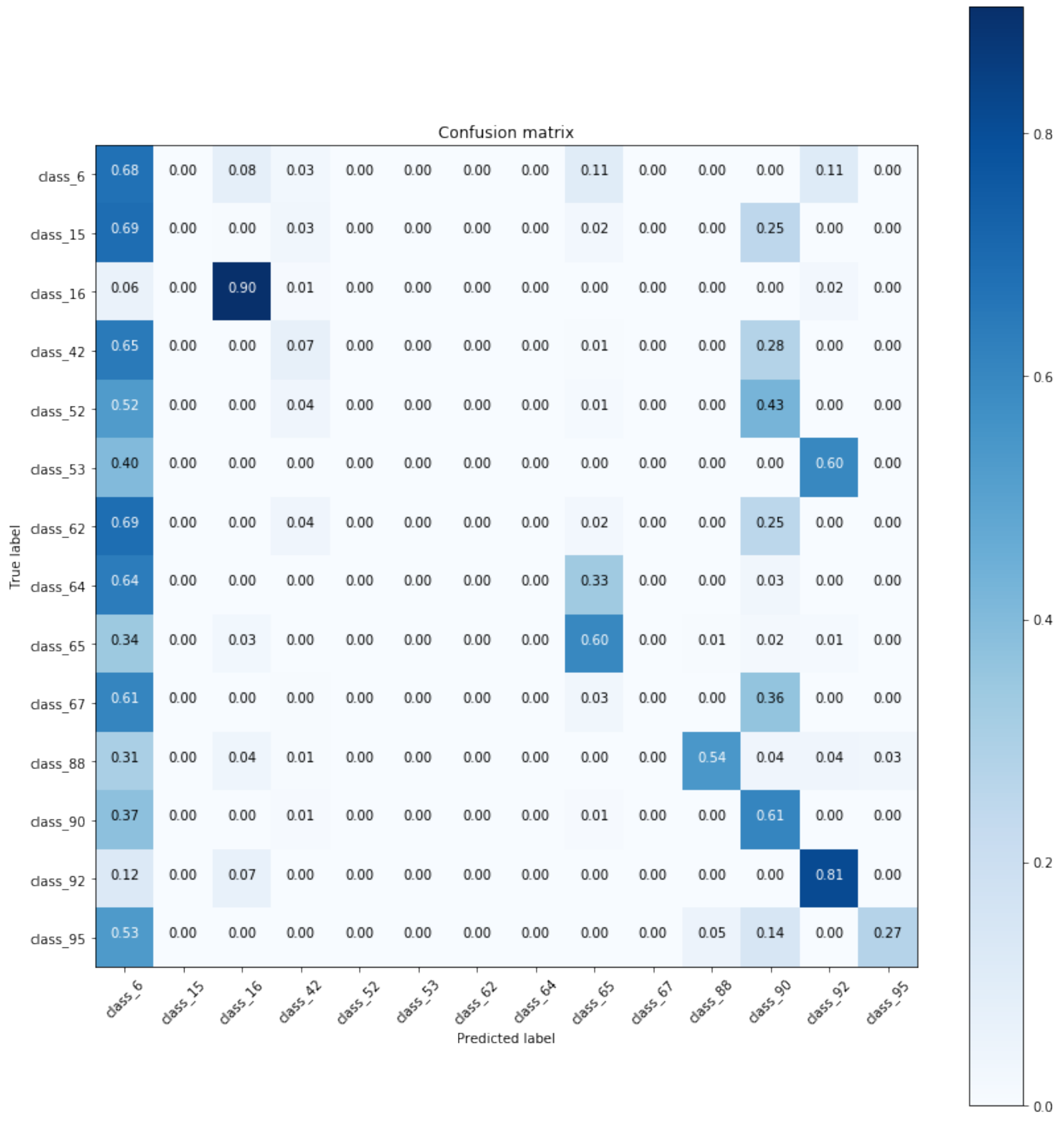


Figure 5.2: Log-loss as a function of the hyperparameter k.

$k = 3$ is chosen for which the log-loss metric is 23.05987. As we see later, this metric is less than 1 for both of the other classifiers. The terrible performance of KNN can be explained by the high dimension of the data – this causes the Euclidean to be less representative and is called the *curse of dimensionality* [20].

The confusion matrix, based on the out-of-fold predictions, is provided in Figure 5.3. It can be seen that the classifier only does a good job of predicting for a few classes and has almost zero accuracy for majority of the object classes. This explains the high log-loss loss since it discriminates against classifiers that have any low accuracy predictions for any particular object class. Interestingly, this KNN implementation wants to classify most objects as class 6.



	class_6	class_15	class_16	class_42	class_52	class_53	class_62	class_64	class_65	class_67	class_88	class_90	class_92	class_95
class_6	0.68	0.00	0.08	0.03	0.00	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.11	0.00
class_15	0.69	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.25	0.00	0.00
class_16	0.06	0.00	0.90	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00
class_42	0.65	0.00	0.00	0.07	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.28	0.00	0.00
class_52	0.52	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.43	0.00	0.00
class_53	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.60	0.00
class_62	0.69	0.00	0.00	0.04	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.25	0.00	0.00
class_64	0.64	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.00	0.00	0.03	0.00	0.00
class_65	0.34	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.60	0.00	0.01	0.02	0.01	0.00
class_67	0.61	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.36	0.00	0.00
class_88	0.31	0.00	0.04	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.54	0.04	0.04	0.03
class_90	0.37	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.61	0.00	0.00
class_92	0.12	0.00	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.81	0.00
class_95	0.53	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.14	0.00	0.27

Figure 5.3: Confusion matrix for the KNN.

To better visualize how KNN works for our dataset, let us consider a scatter plot restricted to our two most successfully predicted object classes, 16 and 92. We also need to choose two out of the 28 features in order to produce a 2D scatter plot. This is done here by comparing feature importances based on Section 5.2.3.1. I pick the second and third most important features, `flux_mean` and `flux_std` because values for `hostgal_photoz` have a significantly higher variance for class 92 objects than for class 16. The resulting scatter plot is shown in Figure 5.4.

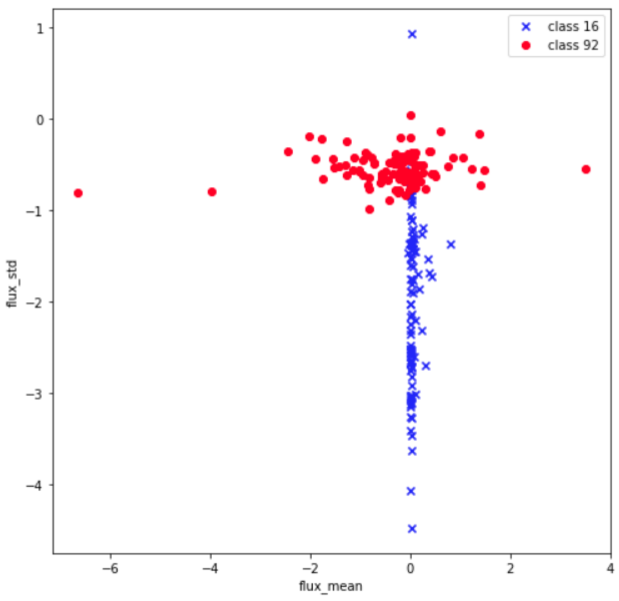


Figure 5.4: Scatter plot of 100 class 92 objects and 100 class 16 objects as a function of the mean and standard deviation of the flux. Expect for a few outliers, the two different object classes form clearly separate clusters, which KNN can take advantage of. One can imagine however, that the clustering will not be as clear for the majority of the objects, explaining poor performance.

5.2.2 Neural Network

In order to simplify things, let us look at a single layer NN with one output neuron as illustrated in Figure 5.5. At the output node, we have an activation function $\phi(z)$ which takes as it's argument $z = \mathbf{x}^T\mathbf{w}$, a linear combination of the input vector $\mathbf{x}$ and the weight coefficients $\mathbf{w}$. In this example, the output $\phi(z)$ is used as the probabilistic classification $\hat{y}$.

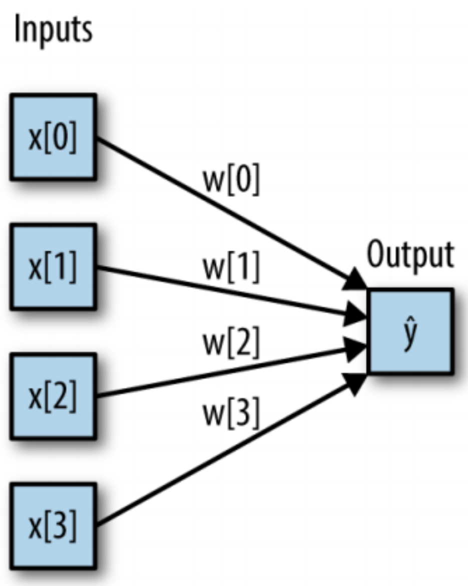


Figure 5.5: Architecture of a single layer neural network with one output node. Source: [9]

We can generalize this example by having n input nodes and m output nodes, in which case we need multiple m weight vectors $\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_m$. We can use those as columns for a $n \times m$ weight matrix. Then the output of that layer becomes $\phi(\mathbf{z})$, where $\mathbf{z} = \mathbf{x}^T\mathbf{W}$. This can be further generalized by adding layers. For example, with three layers we would have three weight matrices $\mathbf{W}_1$, $\mathbf{W}_2$ and $\mathbf{W}_3$.

NN has many hyperparameters, mainly the number of layers and the number of neurons in each layer. As a result it is much harder to pick the optimal values as it was for KNN. Inspired by the Kaggle discussion board, namely [7], I chose a NN with 4 hidden layers, containing respectively 512 neurons, 256 neurons, 128 neurons and 64 neurons. This is an attempt to get the bias as low as possible while still having a reasonable training time [1]. Powers of 2 are used for computational efficiency. For our activation function we chose the popular ReLu function

$$\phi(\mathbf{z})_i = {z_i}^+ = \max(0, z_i), \tag{5.1}$$

which is used in all layers expect the output. In the output layer we use the Softmax activation function

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{5.2}$$

which makes sure that the output is a probabilistic classification vector. Non-linear activation functions allows our NN to capture more indicative feature combinations. The architecture of my NN is summarized in Figure 5.6.

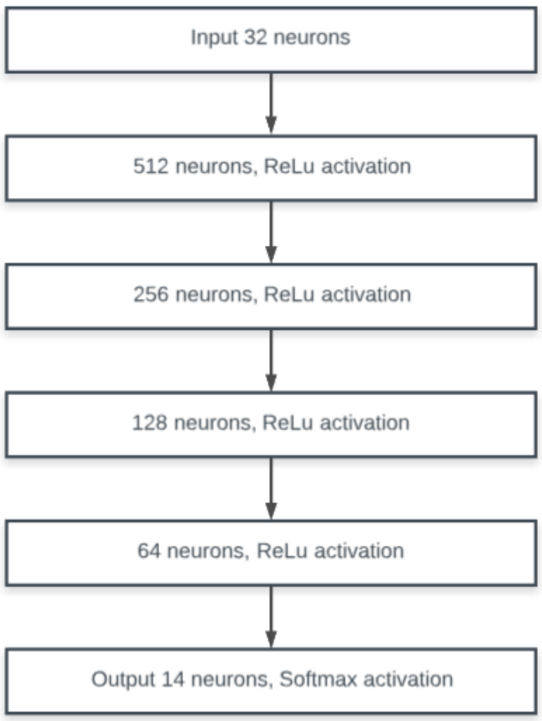

Input 32 neurons
512 neurons, ReLu activation
256 neurons, ReLu activation
128 neurons, ReLu activation
64 neurons, ReLu activation
Output 14 neurons, Softmax activation

Figure 5.6: The architecture of my NN. One rectangle denotes a single layer and an arrow denotes that the two layers are fully connected.

After determining the architecture of a NN, a training process is carried out. The goal is to find parameter values, i.e. weights, for which the loss function is optimized. Here I use the same loss function as (2.3), with weights taken to be the same for all classes (since the actual values are unknown to PLAsTiCC participants) i.e.

$$Q_n = -\frac{1}{14} \sum_{m=1}^{M} \tau_{n,m} \ln \left[p\left(m | d_n \right) \right]. \tag{5.3}$$

[1] My NN took roughly an hour to train.

Optimization is done using a gradient descent algorithm. I chose the widely used `Adam` method for this task [14]. The gradients of the loss function are calculated using a process called backpropagation, which boils down to the chain rule for partial derivatives.

Note that we did not have a training process for KNN. This is because KNN is a so-called *lazy learner* – it just memorizes the training data in order to make predictions [20].

To implement the training of the NN, I used the high-level library `Keras`. The resulting confusion matrix is provided in Figure 5.7. Vast improvements over KNN can be observed. Most classes are predicted correctly the majority of times and there is no obvious bias towards one class like there was for class 6 in the case of KNN. This aligns with the goals of PLAsTiCC much better. The log-loss score obtained by the out-of-fold predictions was 0.95512.

One problem with the NN approach is that there is no obvious way of visualizing what the algorithm is doing, like the scatter plot for KNN and the tree plot for LightGBM.

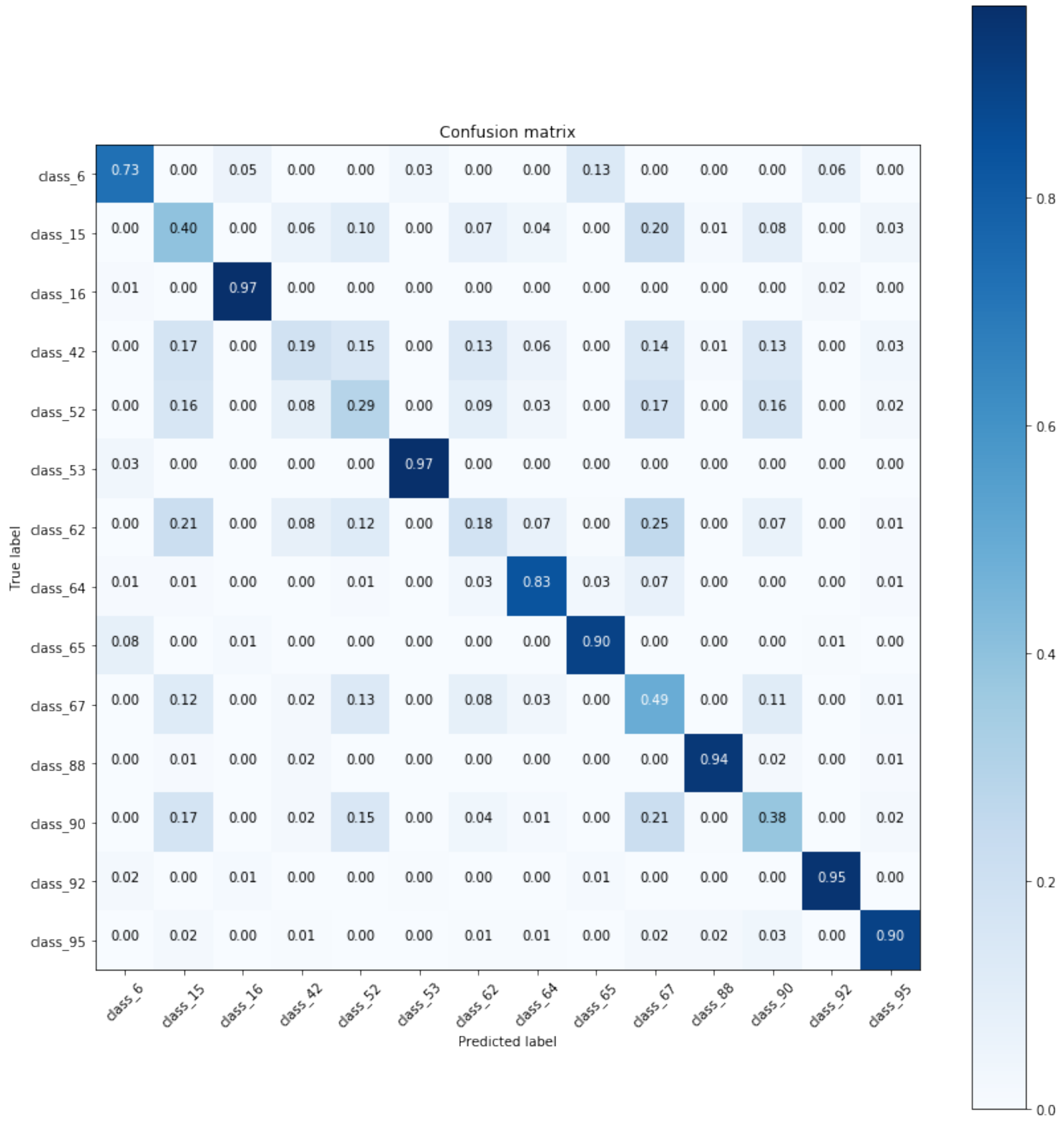


Figure 5.7: Confusion matrix for the NN.

5.2.3 LightGBM

LightGBM is a gradient boosting framework based on decision trees [17]. A decision tree is illustrated in Figure 5.8. In gradient boosting, shallow trees are built in a serial manner where each next tree tries to improve on the mistakes of the previous one [9]. The splits of the trees are chosen so as to optimize the decrease in the expected value of the loss function. For the sake of consistency, I will use the same loss function as for the NN here, given by (5.3).

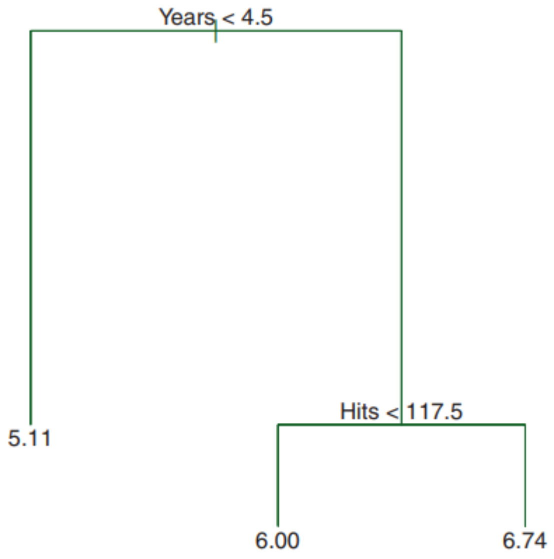


Figure 5.8: A shallow decision tree estimating the log salary of a baseball player based on how many years he has played in the major league and how many hits he made last year. The label at a node indicates the left-hand branch. Source: [11]

Hyperparameter choices for LightGBM were inspired by [6]. In particular `n_estimators` was taken to be 1000. This means that for each of the 14 possible object classes we have 1000 decision trees in the series i.e. 14,000 trees in total[2]. Also, `max_depth` was chosen to be 3, meaning that no tree in this model has more than 8 leaf nodes. An example of one of the trees in the series is shown in Figure 5.9.

There are multiple other gradient boosting frameworks in use by the scientific community, like XGBoost and CatBoost, but LightGBM is faster while not sacrificing a lot of accuracy. This is why it is so popular in many Kaggle competitions including this PLAsTiCC. Indeed, my LightGBM model took a minute to train while the NN model took roughly an hour, while both achieved similar log-loss metrics.

The resulting confusion matrix is shown in Figure 5.10. Again this is a clear improvement over KNN. Comparing with Figure 5.7, we can see that NN performs better for classes 52, 53, 67, while LightGBM performs better for classes 6, 15, 42, 64, 90. This contrast in prediction results suggests that stacking these two models (as was done by many of the Kaggle top teams) would improve performance even further. This is looked at in the next subsection.

[2]The trees appear in an alternating fashion. A tree for estimating the probability of the first class is followed by a tree for the second class which is followed by a tree for the third class etc.

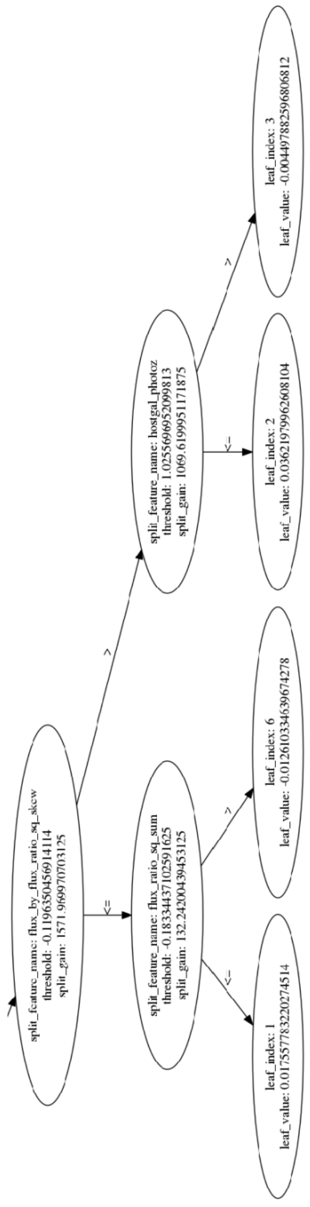

split_feature_name: flux_by_flux_ratio_sq_skew
threshold: -0.1196350456914114
split_gain: 1571.969970703125
<=
>
split_feature_name: flux_ratio_sq_sum
threshold: -0.18334437102591625
split_gain: 132.24200439453125
split_feature_name: hostgal_photoz
threshold: 1.0255696952099813
split_gain: 1069.6199951171875
<=
>
<=
>
leaf_index: 1
leaf_value: 0.017557783220274514
leaf_index: 6
leaf_value: -0.012610334639674278
leaf_index: 2
leaf_value: 0.036219799626081 04
leaf_index: 3
leaf_value: -0.004497882596806812

Figure 5.9: An example of a decision tree in our LightGBM model. Only half of the tree is shown – the whole tree has 8 leaf nodes. This is the 54th in the series of 14,000 trees used in the model and estimates the probability of the object being in class 64. `split_feature_name` denotes what feature is used at that split. `split_gain` tells us how much the expected value of the loss function is decreased by that split. We can use the logistic function, $f(x) = \frac{1}{1+e^{-x}}$ to get the probability from `leaf_value`.

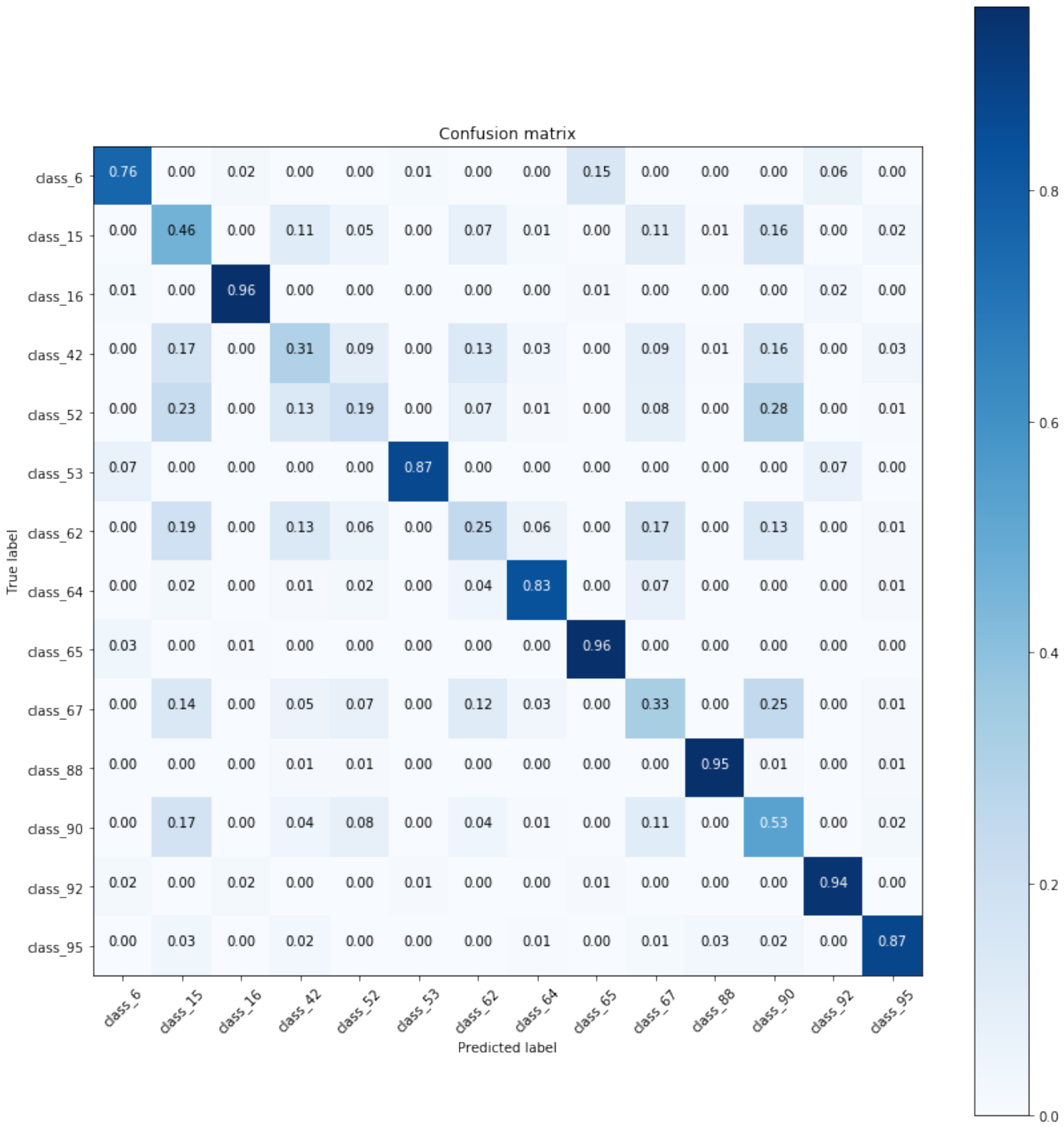


Figure 5.10: Confusion matrix for the LightGBM.

5.2.3.1 Importance of features

As explained in Figure 5.9, for each tree in the model and for each split we have a `split_feature_name` and a `split_gain`. Using these, a metric can be constructed to measure the importance of a feature i.e. how much information it contains. To do this, we look at all the splits where that feature was used and then add up their `split_gain` values. The LightGBM framework has an in-built function for this called `feature_importances_`. A plot with the importance metrics for our features is shown in Figure 5.11.

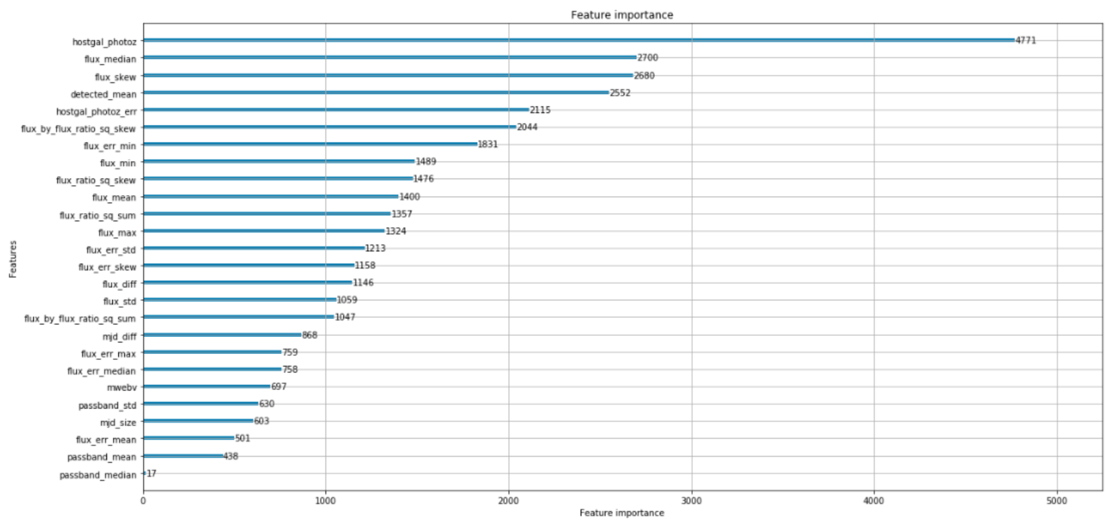


Figure 5.11: Features and importance metrics based on the LightGBM model.

5.2.4 Stacking and results

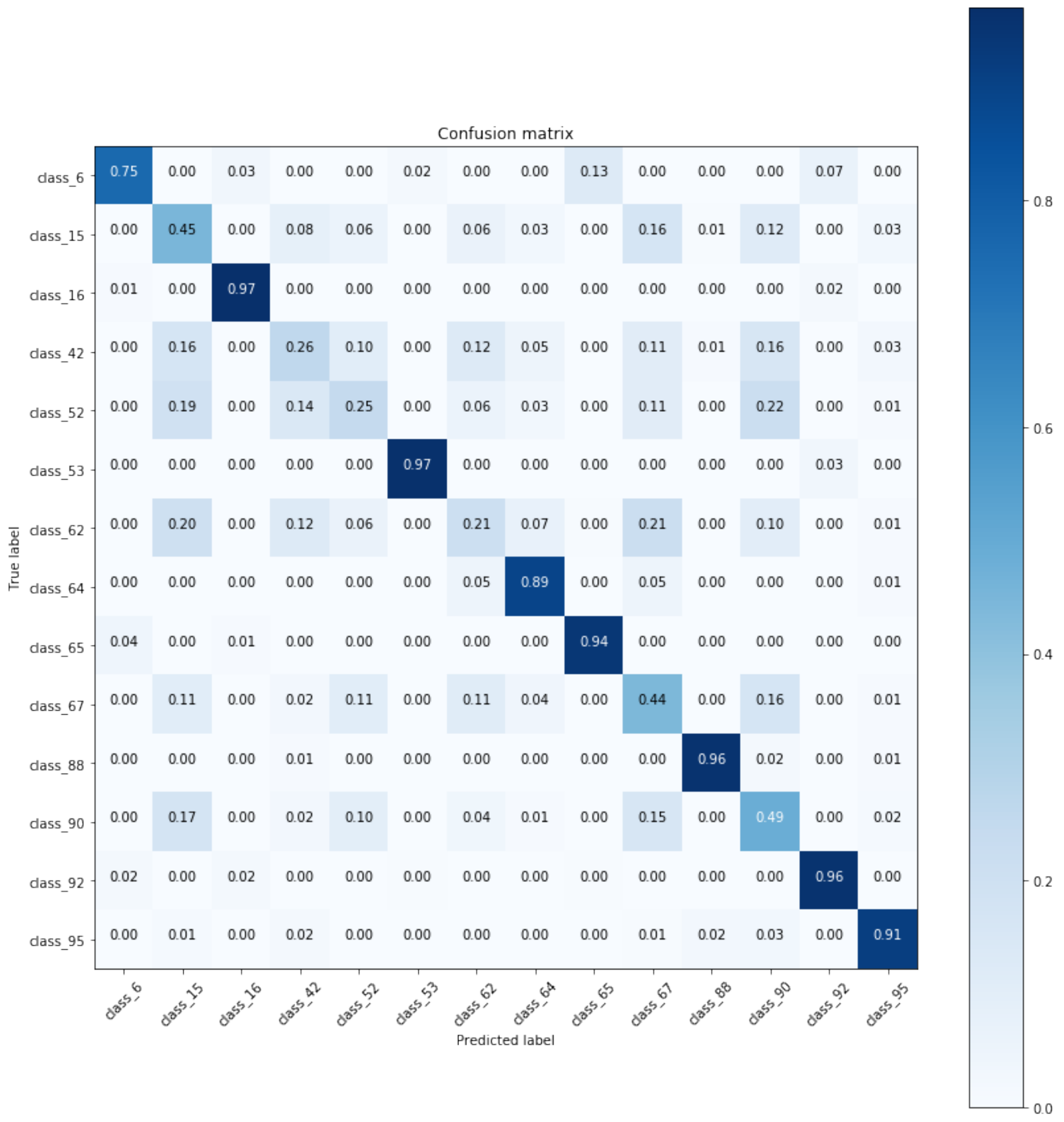
Stacking is carried out by averaging the prediction matrices of different methods. The first stack comprises of the two most successful methods NN and LightGBM. The second stack combines all three models. In other words, denoting the probabilistic classification by $\mathbf{y}$, the results of stacking would be

$$\mathbf{y}_{S1} = \frac{\mathbf{y}_{NN} + \mathbf{y}_{LightGBM}}{2} \tag{5.4}$$

and

$$\mathbf{y}_{S2} = \frac{\mathbf{y}_{KNN} + \mathbf{y}_{NN} + \mathbf{y}_{LightGBM}}{3}. \tag{5.5}$$

The Confusion matrices of the two stacks are shown in Figures 5.12 and 5.13.



True label \ Predicted label	class_6	class_15	class_16	class_42	class_52	class_53	class_62	class_64	class_65	class_67	class_88	class_90	class_92	class_95
class_6	0.75	0.00	0.03	0.00	0.00	0.02	0.00	0.00	0.13	0.00	0.00	0.00	0.07	0.00
class_15	0.00	0.45	0.00	0.08	0.06	0.00	0.06	0.03	0.00	0.16	0.01	0.12	0.00	0.03
class_16	0.01	0.00	0.97	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00
class_42	0.00	0.16	0.00	0.26	0.10	0.00	0.12	0.05	0.00	0.11	0.01	0.16	0.00	0.03
class_52	0.00	0.19	0.00	0.14	0.25	0.00	0.06	0.03	0.00	0.11	0.00	0.22	0.00	0.01
class_53	0.00	0.00	0.00	0.00	0.00	0.97	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00
class_62	0.00	0.20	0.00	0.12	0.06	0.00	0.21	0.07	0.00	0.21	0.00	0.10	0.00	0.01
class_64	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.89	0.00	0.05	0.00	0.00	0.00	0.01
class_65	0.04	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.94	0.00	0.00	0.00	0.00	0.00
class_67	0.00	0.11	0.00	0.02	0.11	0.00	0.11	0.04	0.00	0.44	0.00	0.16	0.00	0.01
class_88	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.96	0.02	0.00	0.01
class_90	0.00	0.17	0.00	0.02	0.10	0.00	0.04	0.01	0.00	0.15	0.00	0.49	0.00	0.02
class_92	0.02	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.96	0.00
class_95	0.00	0.01	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.03	0.00	0.91

Figure 5.12: Confusion matrix for the stack of NN and LightGBM. Stacking has made classification more accurate and gotten rid of some of the noise.

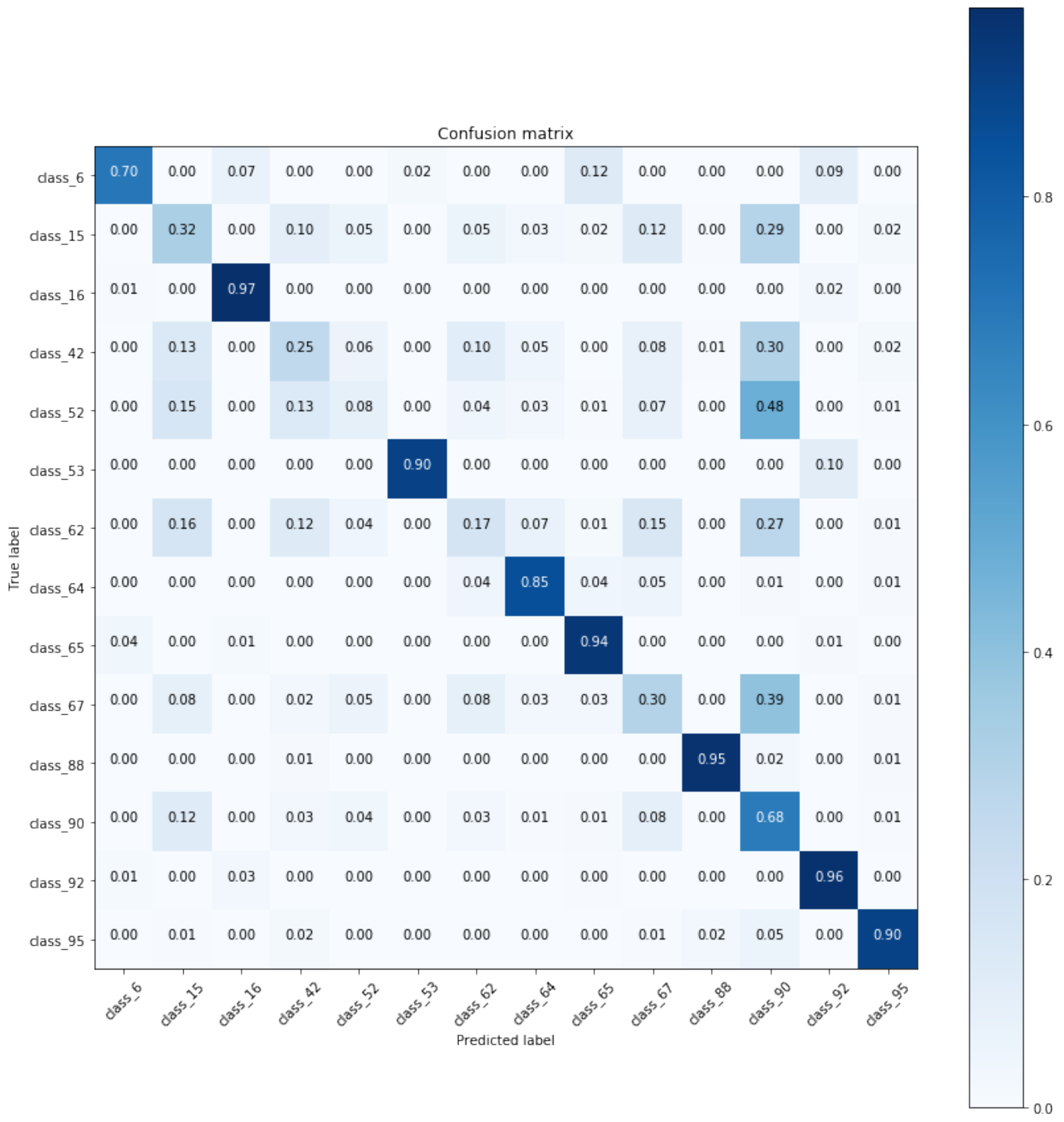


Figure 5.13: Confusion matrix for the stack of all three algorithms. Averaging the predictions has gotten rid of the KNN bias towards class 6. On the other hand it has created a new (smaller) bias towards class 90.

5.3 Discussion

The results of stacking are compared with the results of individual methods in Table 5.2.

Clearly, adding KNN to the stack was not useful. This is probably because the insight gathered by KNN was rudimentary enough that both LightGBM and NN also captured it. Thus KNN cannot provide any new information to the stack.

Interestingly, NN performs worse than LightGBM on out-of-fold predictions but better on test set. This might be explained by the training set not being representative. It is possible that the NN uses the feature in such a way that the light-curves do not have to be as well-sampled. This would make it a more useful algorithm for the LSST.

	KNN	**NN**	**LightGBM**	**NN & LightGBM**	**All three**
OOF	23.05987	0.95512	0.93377	0.88307	1.10424
Test		1.35782	1.38951	1.30448	

Table 5.2: Performance of the various ML algorithms and stacks on out-of-fold predictions and test predictions. Note that KNN was not run on the test dataset because the time it takes for it to make a single classification is significantly longer than for NN and LightGBM. For each new object it has to find the k training objects "closest to it" which is computationally costlier than running the input through a pre-built decision tree or NN.

As opposed to [15], I did not find there to be a major difference in results between the NN and decision tree based approaches. This might be because I used a multi-layer NN which can include more complicated non-linear feature combinations. In fact, the NN showed the best performance on the less-sampled test dataset. This agrees with a recent paper by Muthukrishna et al. [18] which proposes an early time-series classification tool for the LSST, also based on a neural network and performing well on light-curves with limited phase coverage.

The astronomical phenomena corresponding to the object classes were revealed in [12]. These can be used to discuss the real-life implications of my best performing classifier, the NN and LightGBM stack. Based on Figure 5.12, we can find that it was very accurate (more than 90% of objects classified correctly) for Eclipsing Binary stars, Pulsating variable stars, M-dwarf stellar flares, Active Galactic Nuclei, RR Lyrae and Super-Luminous supernovae (hereafter SN). Performance was poor (less than 30% of objects classified correctly) for Type II SN, Type Ibc SN and Type Iax SN. These were most often confused with other types of SN, mainly Type Ia-91bg and Type Ia, but also with Tidal Disruption Events. This agrees with the results of [15] – all their ML classifiers had trouble differentiating between different Type I supernovae.

It is likely that performance of the classifiers for different supernovae would be improved by including Gaussian processes in the feature extraction step as in Kyle Boone's solution.

My best performing method attained a weighted log-loss metric of 1.30448 which would have placed me on the 554th place in the competition which is roughly in the middle. It is important to note that my solution only used insight from Kaggle discussion that was public during the competition, mainly [6] and [7]. Attaining better results would require heavy trial-and-error in feature extraction[3], curve interpolation[4] and possibly a cleverer stacking method.

It would of course be easy to attain a better log-loss metric retrospectively. For example, one can just average the prediction matrices of the top 5 teams as discussed in Section 4.

[3]All the top 5 teams had over 100 submissions. To compare, just attaining a single prediction table based on the 20 Gigabyte training set took me 2 hours of computation time.

[4]Also computationally expensive as discussed above.

6 Conclusion

Out of the three ML classifiers and their stacks, the best performing one was the combination of NN and LightGBM. The only objects it had trouble classifying were different types of supernovae.

Analysing the approaches of the top 5 teams of PLAsTiCC gave insight into what makes a successful classifier for the challenge. To improve on the best classifier proposed here, while possibly losing in computation time, both feature extraction and the ML algorithm could be made more complex. Feature extraction can benefit from light-curve interpolation using Gaussian processes. The ML stack would perform better if we added more base models, for example more LightGBM and NN with different hyperparameters. One could also make the actual stacking more complex than simply an arithmetic mean of the base models, like training a new ML classifier that takes base model outputs as input.

On the other hand, dropping less useful feature could make the pipeline more lightweight which is an important aspect considering the massive volume of data that will come from the LSST. These features could be determined using the feature importance metrics from LightGBM.

Many of the results of PLAsTiCC were further confirmed by reviewing an earlier similar challenge, the SPCC.

Bibliography

[1] Overview of 1st place solution by Kyle Boone, . URL https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75033.

[2] Overview of 2nd place solution by team Mike and Silogram, . URL https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75059.

[3] Overview of 3rd place solution by team Major Tom, . URL https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75116.

[4] Overview of 4th place solution by Ahmet Erdem, . URL https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75011.

[5] Overview of 5th place solution by team SKZ lost in translation, . URL https://www.kaggle.com/c/PLAsTiCC-2018/discussion/75050.

[6] PLAsTiCC in a kernel meta and data. URL https://www.kaggle.com/ogrellier/plasticc-in-a-kernel-meta-and-data.

[7] Simple Neural Net for Time Series Classification. URL https://www.kaggle.com/meaninglesslives/simple-neural-net-for-time-series-classification.

[8] N. S. Altman. An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician*, 46(3):175, 1992. doi: 10.2307/2685209.

[9] Muller Andreas C. and Sarah Guido. *Introduction to machine learning with Python: a guide for data scientists.* OReilly, 2017.

[10] J. Guy, P. Astier, S. Baumont, D. Hardin, R. Pain, N. Regnault, S. Basa, R. G. Carlberg, A. Conley, S. Fabbro, D. Fouchez, I. M. Hook, D. A. Howell, K. Perrett, C. J. Pritchet, J. Rich, M. Sullivan, P. Antilogus, E. Aubourg, G. Bazin, J. Bronder, M. Filiol, N. Palanque-Delabrouille, P. Ripoche, and V. Ruhlmann-Kleider. SALT2: using distant supernovae to improve the use of Type Ia supernovae as distance indicators.

[11] Gareth James, Daniela Witten, Trevor J. Hastie, and Robert J. Tibshirani. *An introduction to statistical learning: with applications in R.* Springer, 2017.

[12] R. Kessler, G. Narayan, A. Avelino, E. Bachelet, R. Biswas, P. J. Brown, D. F. Chernoff, A. J. Connolly, M. Dai, S. Daniel, R. Di Stefano, M. R. Drout, L. Galbany, S. Gonzlez-Gaitn, M. L. Graham, R. Hloek, E. E. O. Ishida, J. Guillochon, S. W. Jha, D. O. Jones, K. S. Mandel, D. Muthukrishna, A. O'Grady, C. M. Peters, J. R. Pierel, K. A. Ponder, A. Pra, S. Rodney, and V. A. Villar. Models and Simulations for the Photometric LSST Astronomical Time Series Classification Challenge (PLAsTiCC). 2019.

[13] Richard Kessler, Bruce Bassett, Pavel Belov, Vasudha Bhatnagar, Heather Campbell, Alex Conley, Joshua A. Frieman, Alexandre Glazov, Santiago Gonzalez-Gaitan, Renee Hlozek, Saurabh Jha, Stephen Kuhlmann, Martin Kunz, Hubert Lampeitl, Ashish Mahabal, James Newling, Robert C. Nichol, David Parkinson, Ninan Sajeeth Philip, Dovi Poznanski, Joseph W. Richards, Steven A. Rodney, Masao Sako, Donald P. Schneider,

Mathew Smith, Maximilian Stritzinger, and Melvin Varughese. Results from the Supernova Photometric Classification Challenge. 2010. doi: 10.1086/657607.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. 2014.

[15] Michelle Lochner, Jason D. Mcewen, Hiranya V. Peiris, Ofer Lahav, and Max K. Winter. Photometric Supernova Classification With Machine Learning. *The Astrophysical Journal Supplement Series*, 225(2):31, 2016. doi: 10.3847/0067-0049/225/2/31.

[16] Alex Malz, Rene Hloek, Tarek Allam Jr, Anita Bahmanyar, Rahul Biswas, Mi Dai, Llus Galbany, Emille Ishida, Saurabh Jha, David Jones, Rick Kessler, Michelle Lochner, Ashish Mahabal, Kaisey Mandel, Rafael Martnez-Galarza, Jason McEwen, Daniel Muthukrishna, Gautham Narayan, Hiranya Peiris, Christina Peters, Christian Setzer, The LSST Dark Energy Science Collaboration, The LSST Transients, and Variable Stars Science Collaboration. The Photometric LSST Astronomical Time-series Classification Challenge (PLAsTiCC): Selection of a performance metric for classification probabilities balancing diverse science goals. 2018.

[17] Microsoft. Microsoft/LightGBM, Apr 2019. URL `https://github.com/Microsoft/LightGBM`.

[18] Daniel Muthukrishna, Gautham Narayan, Kaisey S. Mandel, Rahul Biswas, and Rene Hloek. RAPID: Early Classification of Explosive Transients using Deep Learning. 2019.

[19] Gautham Narayan, Tayeb Zaidi, Monika D. Soraisam, Zhe Wang, Michelle Lochner, Thomas Matheson, Abhijit Saha, Shuo Yang, Zhenge Zhao, John Kececioglu, Carlos Scheidegger, Richard T. Snodgrass, Tim Axelrod, Tim Jenness, Robert S. Maier, Stephen T. Ridgway, Robert L. Seaman, Eric Michael Evans, Navdeep Singh, Clark Taylor, Jackson Toeniskoetter, Eric Welch, and Songzhe Zhu. Machine Learning-based Brokers for Real-time Classification of the LSST Alert Stream. 2018. doi: 10.3847/1538-4365/aab781.

[20] Sebastian Raschka and Vahid Mirjalili. *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow*. Packt Publishing, 2017.

[21] The PLAsTiCC team, Tarek Allam Jr., Anita Bahmanyar, Rahul Biswas, Mi Dai, Llus Galbany, Rene Hloek, Emille E. O. Ishida, Saurabh W. Jha, David O. Jones, Richard Kessler, Michelle Lochner, Ashish A. Mahabal, Alex I. Malz, Kaisey S. Mandel, Juan Rafael Martnez-Galarza, Jason D. McEwen, Daniel Muthukrishna, Gautham Narayan, Hiranya Peiris, Christina M. Peters, Kara Ponder, Christian N. Setzer, The LSST Dark Energy Science Collaboration, The LSST Transients, and Variable Stars Science Collaboration. The Photometric LSST Astronomical Time-series Classification Challenge (PLAsTiCC): Data set. 2018.