CS342 Assignment 2 Nicola Branchini, ID: 1614075

Abstract

This competition involves a particularly challenging classification of light curves. Domain knowledge is particularly useful to extract meaningful features, such as periodicity for galactic objects, in order to distinguish the most similar classes. In summary, the most important factor in this competition should be good feature engineering, in particular specific to light curves, followed by sound hyperparameter optimization and model ensembling. In particular, we think that dealing better with class imbalances with techniques such as **SMOTE** would have improved performance further.

Data Exploration, Feature Engineering, & Time Series Analysis

Task 1

The light curves in the dataset can be divided into two kinds: galactic and extragalactic . The former are closer, exhibit periodicity and have less measurements uncertainty, therefore are likely to be easier to classify than the latter. There are precisely 390,510 galactic objects and 3,102,380 extragalactic objects in the test set, whereas the training set contains 2325 galactic and 5523 extragalactic. This gives percentages of galactic objects of ≈ 11.2% and ≈ 29.6% for test and train set respectively. These two categories are fully disjoint (as shown in figure 1), hence given the redshift of an object we can immediately rule out a lot of options for its classification (as redshift exactly equal to 0.0 corresponds to galactic). Extra-galactic objects are harder to classify also because most of these are supernovas, which differ among them only by a few factors such as rate of decay. This is in contrast to galactic objects, which as said are periodic and in general exhibit more distinctive features. Overall, this implies that a good classifier will be one that performs relatively well at distinguishing extragalactic objects.

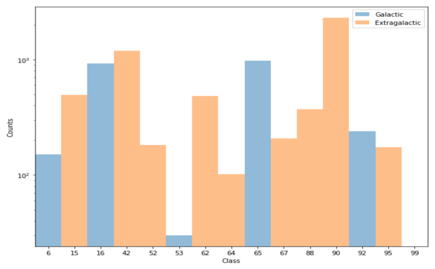
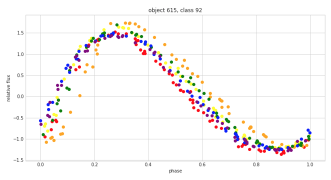
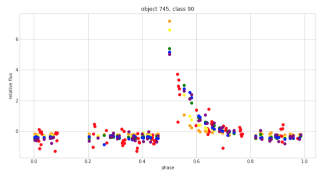


Figure 1: Plot of class against number of objects, galactic in blue. No classes have both galactic and extragalactic objects.

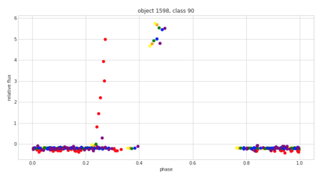
The periodicity of galactic objects can be tested by plotting light curves of different classes, as exemplified in figure 2. The time series of this dataset are particularly challenging to classify for several reasons. These are irregular in terms of observation intervals, unsynchronized across different passbands and some of the uncertainties in redshift values in the test set are catastrophically wrong, according to the paper describing the challenge [9]. On the whole, major challenges are: observations even in the same band are not evenly spaced, timeseries lengths vary significantly (which makes techniques such as CNNs harder to apply and increase computational time if padding is used), and as time series for different objects are not synchronised, it is reasonable to believe that a good classifier would need to discover features that are invariant with time shifting and stretching at least to some extent.
We know that some classes of objects can exhibit periodicity (only galactic classes have this property): in order to discover the period of the time series, we can normalise all series and then extract features about the most prominent frequencies / periods (this has been done in [5]). As shown in figure 2, object from different classes can have different kinds of periods. For example, object 615 from class 92 is periodically changing its magnitude, while objects 745 and 1598 periodicities (both class 90) are more stationary most of the time with sudden bursts.



(a) Object 615, class 92



(b) Object 745, class 90



(c) Object 1598, class 90

Figure 2: Phase space plots. Phase against relative flux.

The emerging pattern here is that it seems that if bands agree in frequency, an object is likely to have periodic behaviours. Calculated periods alone cannot tell us the full picture, as there are plenty of objects in the dataset without clear periodic patterns. However, determining whether an object has a periodic pattern appears to be potentially useful in determining the object class. As explained later for task 2, we calculated time series features in order to attempt to extract this kind of information.

Task 2

Firstly, we remove some features from the raw data that appear do not appear to be useful from both the analysis of the problem and from results of the experiments. These are `ra`, `decl`, `gal_l`, `gal_b` and contain information about the galactic coordinates and position, which should not affect classification. In addition, we also decided to drop `hostgal_specz`: even if this is a far more accurate measure of redshift than `hostgal_photoz`, it is not present in the test set. One could think of techniques that attempt to predict this and use it, but this would be hard to achieve successfully.
After this, we split the data into two disjoint subsets, galactic and extragalactic and train separate models. We then calculate the amplitude of objects' light as described in the lecture. Specifically, this feature is computed in the following way, for each row in the raw data (train at training time, test at prediction): $A = -2.5 \cdot \log(flux) - distmod$. In order to preserve relative order, we take the absolute value of the flux and negate the final result. The `distmod` Not A Number values are naturally turned into zeros. At this point, time series features along with aggregations are computed (always for each passband, to preserve information). Specifically, we used [6] to compute
A very smart and powerful feature according to a discussion in public kernels ([7]) appears to be useful to separate "one event" objects such as supernovae from "cyclic event" objects such as cepheids. This is computed as $mjd\_diff = \max(mjd) - \min(mjd)$. Other powerhful features have proved to be (will be detailed in the progression graph section) specific time series values such as FFT coefficients, which help to extract periodicity information, skewness (representing the symmetry of the probability density function of the amplitude of a time series, hence potentially helpful with some supernovas) and kurtosis (measures the peakedness of the PDF of a time series, perhaps less useful than the previous as we can already distinguish well between supernovas which usually have high peaks)
As far as predictions for class 99 are concerned, we simply take the probability of the sample not being each class and multiply them: $p(99) = \prod_c (1 - p(c))$. Approaches such as outlier detection with Isolation Forests have been taken into consideration, however from kernel discussions show that other people have tried this approach failing to improve their performance. Some kernels scale $p(99)$ we calculated by multiplying by 0.14 and dividing by the mean value, however we did not find justification for this therefore it has not been implemented.
We also found in the following Kaggle discussion from a comment of the currently 2^{nd} Kaggler in the competition that it is possible to derive a better measurement of `distmod` [1], which is a very important feature as it was mentioned in the guest lecture and confirmed by LGBM's feature importances (described later). It was not possible to test whether precisely recalculating the absolute magnitude with this less noisy `distmod` measurement gives a performance improvement, however this is one of the features used in the submission which got the best score with LGBM (that is, we cannot determine its impact as other changes were made). This is shown in the progression graph 4.

Task 3

For data augmentation we first considered the approaches described in the article linked in the assignment. The techniques presented are **window slicing** and **window warping**. These are both arguably not suited for the specific problem of this competition: the former would extrapolate sections from the time series but this would misclassify most objects as the series are sampled at irregular time intervals and a lot of them (extragalactic) are not periodic, while the latter modifies the precise shape of the times series, which again is fundamental to discern distinct classes in this challenge. Therefore, in order to augment the dataset we doubled each point in the training set by adding to its flux value a gaussian noise with standard deviation proportional to `flux_err` (this is because some `flux_err` values are very small). This is a simple approach to implement and only takes a couple hours to run. The performance of the best random forest has not improved with data augmentation (only making it worse by 0.02 local CV, competition multi-weighted logloss). As far MLPs and CNNs are concerned, this approach made them overfit aggressively, scoring more than 10.0 in the leaderboard and giving (incorrect) overconfident predictions in most rows during test time (that is, the prediction matrix had a high number of rows with a 1 probability for a specific class and 0s for all others).
After this had been implemented, an alternative technique had been proposed in seminars. This consists in calculating the changes between consecutive points, calculate the standard deviation of these and use it as noise in the same way as we previously used `flux_err`. However we decided to not implement this as any of these simple data augmentation approach did not seem to benefit any of the classifiers used, especially MLPs, instead causing them to overfit more aggressively.

Machine Learning Models: Artificial Neural Networks & Deep Learning

Task 4

Random Forest with raw time series values

For the first submission, a RF (Random Forest) has been implemented with simple aggregations over `flux`, `flux_err`, `mjd` and `detected` such as mean, median, stdev (etc..). These were also obtained for each passband in order to retain this information, which would be lost by aggregating over all passbands. The parameters of the random forest are selected by GSCV (GridSearch Cross-Validation) with 5 folds, initially according to negative log-loss, as it is the metric that most resembles the loss used in the competition (the more appropriate metric is used for subsequent submissions). For this submission data was not split into galactic/extragalactic. A parameter that has not been gridsearched with random forests across models is `n_estimators`, as it is beneficial to have as many as possible subject to computational time. For the first submission 2000 estimators were used. The depth of the trees selected by GSCV controls overfitting. It is not trivial to have intuition on which would be an appropriate set of values for tree depths, so values in `np.logspace` have been used. This first submission scored 2.625 as shown in the progression graph 4. This is slightly less than the naive benchmark which means that the model has not been able to perfectly identify light curves as extra/intra galactic. Standard scaling was not applied for training or prediction as it is irrelevant to tree-based estimators.

MLP with raw time series values

The MLPs have implemented has been using Keras classifiers. It would be possible to GS the number of epochs and batch size with the `sklearn` wrapper, but it was not considered necessary as Keras allows to use model checkpoints (i.e. saving the best model over epochs) as well as early stopping. It is therefore not feasible to gridsearch automatically the immense number of hyperparameters that compose a Keras sequential model (number of layers, units per layer, activations, optimizers, initializers, learning rate, l1/l2 layerwise regularizers, dropouts), therefore we have generally been following practical guidelines suggested in [2]. The approach taken was to train a fairly large network and overfit, in order to later add regularization. The model for first submission has 8 layers with dropout regularization (no batch normalization, no other regularization) on alternate layers of 0.5. We trained this network with early stopping and achieved local validation loss (competition loss) of 1.5. However these losses while not useless as benchmarks clearly do not reflect true LB score, due to the fact that the training set is not representative of the test set as mentioned in [9].
Predictions of the models were inspected before submissions to better understand the relationship between predictions and LB score. It is easy to see that the worst models exhibit overconfident predictions, assigning probability of 1 to a single class for a significant amount of objects. On the contrary, the best models that were tuned (especially RFs and LGBMs) distributed probability mass more uniformely. Because of the high number of hyperparameters, most of the tuning has been done to improve local CV scores, as evaluating the true performance by predicting is very time expensive. Findings during hyperparameters tuning include : batch normalization greatly improves CV scores (this is due to the fact that standardizded data is crucial for neural networks and we can put a batchnorm as first layer), adding l1/l2 layerwise regularization makes the model greatly underfit, performance is best with Xavier weight initialization (as described in [3]). Learning is perhaps the most important hyperparameter and we used a Keras callback to anneal it according to validation scores.
In summary, the first MLP with no galactic split and no feature engineering (except simple functions) achieved 2.233 LB score. The architecture that we settled with consists of 5 hidden layers, starting with 256 units and dividing by two at each layer (powers of two are mostly chosen for neatness), applying dropouts and batch normalization. This result as described later has been improved by FE and splitting the data.

Task 5

Random Forest with FE & data augmentation

The first feature that has been added was the calculation of absolute magnitued as described previously. We also included the competition multi-weighted log loss with weights determined in Kaggle discussions in order to evaluate RF performance on the training set. All the features described in task 2 were used to improve performance. Notably, the greater improvement (both locally and for LB score) comes from the training separate models for galactic/extragalactic. Using RF's feature importances, we discover `hostgal_photoz` (redshift) to be the most important feature, followed by `distmod` and other useful expansions of the flux suggested in several public kernels such as $flux\_ratio\_sq = \left(\frac{flux}{flux\_err}\right)^2$. For the maximum score achieved with a RF, we used the complete feature set amounting to about 160 features along with 20000 estimators and 25 tree depth. This RF scored 1.545 (LB score). It is believed the number of estimators was unnecessarily large and from further experiments and confrontation with other students, it appears that the order of magnitude of 10^3 is enough and that the difference is made the feature engineering approach.

MLP with FE & data augmentation

The same features described in task 2 and in the previous task have been used to improve MLP performance. Again, splitting the data has a great benefit on these relatively weak models, as they are not able to perfectly discover this relationship automatically. The approach to tuning separate models has been unchanged from the previous case, and the same architecture described in task 4 was used for both models, as there were no noticeable improvements by tuning this further. By adding feature engineering approaches described in task 2 and in previous tasks, it was possible to achieve a LB score of 1.735.

Task 6

CNN with data augmentation and raw time series

There are several possible approaches to using a CNN for time series. One could use a 1D CNN where the passband information is kepts as a feature, and could also keep other features that are not specifically one of the three time series available (`flux`, `flux_err`, `detected`). It was decided to use 2D convolutional layers instead, in the hopes of mirroring a multi-channel image of the time series, with the convolutions trying to extract the features that distinguish objects automatically. The input to the CNN is the following: for each object, we select the values of `flux`,`flux_err`,`detected` from its complete time series. We do this for each passband, so that we get 6 matrices per object, each of dimension (`length of time series times`×3). The idea is to model the time series as images and extrapolate local features with CNNs' local kernel convolution. The input to the CNN is shown in figure 3. The approach to the architecture has again been following the guidelines provided in [2]. We used zero padding and no pooling layers as the networks with which we experiment are of relatively small size, so there is no need for downsampling. The weight matrices are initialized with uniform distribution. Weight initialization is important to prevent the model to just being linear. It was found that uniform gave better CV scores than Xavier initialization. We use a kernel size (1, `n_columns`), which effectively slides top to bottom for each row simulating a temporal 1D convolutional layer. We believe the CNN to perform worse than an MLP because, from how it was implemented, it does not exploit the useful feature engineering techniques that are applicable to this problem, and its structure is not advanced enough to discover even a good amount of these automatically. Data augmentation described previously has been applied to the CNN. However, best performance (LB score, corresponding to 2.627) was achieved without augmenting the data. This is likely caused by the naivety of the augmentation technique that makes the network overfit. The architecture used consisted of only one convolutional layer followed by two fully connected layers. An approach that has been found in public kernels consisted of binning the values of flux in discretized `mjd` intervals, `t` . Another approach that was not attempted due to time constraints is to download the best currently performing CNN for image recognition and try to adapt the model to this specific problem. In summary, the CNN was the worst performing model. This is because in this problem feature engineering can be efficiently applied directly, and it is hard to automate this process with convolutional layers.

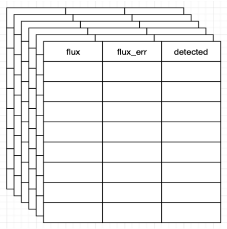


Figure 3: Shape of the data input to the CNN.

Moreover, incorporating `mjd` information is likely important, as removing it implicitely defines unit timesteps which is not accurate. A technique to incorporate these is binning, however due to time constraints this has not been implemented.

Performance Extension

In order to achieve maximum performance an ensemble of gradient-boosted models was used. Specifically, we used Microsoft's LightGBM which is a tree-based gradient-boosted estimator. This approach was chosen as opposed to variants of Recurrent Neural Networks. As mentioned in the abstract, the problem at hand is one with structured data, and many options (primarily time series feature engineering approaches) are available to extract meaningful information. In these problems gradient boosted-based models perform best, as opposed to deep learning ([4]).
Hyperparameters have been optimized by *Bayesian optimization*. Manual tuning and even more robust methods like GSCV are uninformed searches on the hyperparameter space. These methods are just sampling points from the hyperparameters-loss space independently and are not using information from the acquisition of data ([11]). Specifically, the approach was the following: by analysing the best settings on a set number of evaluations, we tried to gain intuition on the role of parameters (also referring to [8]), and created 7 distinct models with different settings. These are chosen so that some

models are more sensible to overfitting while others less. The parameters which we have mostly focussed in tuning are `n_estimators`, `max_depth`, `boosting_type`, `reg_alpha` . We then train each of these models on the full training set , and at prediction time we average the predictions of each model. Therefore, this is effectively a naive ensembling of models with different hyperparameters. Further approaches which were not implemented due to time constraints are data imputation (for example, by downloading the set of best predictions available, selecting the rows were these are very confident on one class, merging with the test set and adding to the training set) and **SMOTE** (**S**ynthetic **M**inority **O**versampling **TE**chnique). It is expected that particularly the latter would give a notable improvement: this technique deals with imbalanced data by oversampling the least represented classes (or undersampling the most represented ones) [10].

Task 7

In this section we show the progression over the course of the competition of the **best** models, that is we only include the best submission score for each approach attempted and corresponding to the different tasks. The best performing model is clearly the ensemble of LGBMs tuned with Bayesian optimization. However, as mentioned before it is clear how smart feature engineering is able to considerably improve simpler models such as RFs and MLPs.

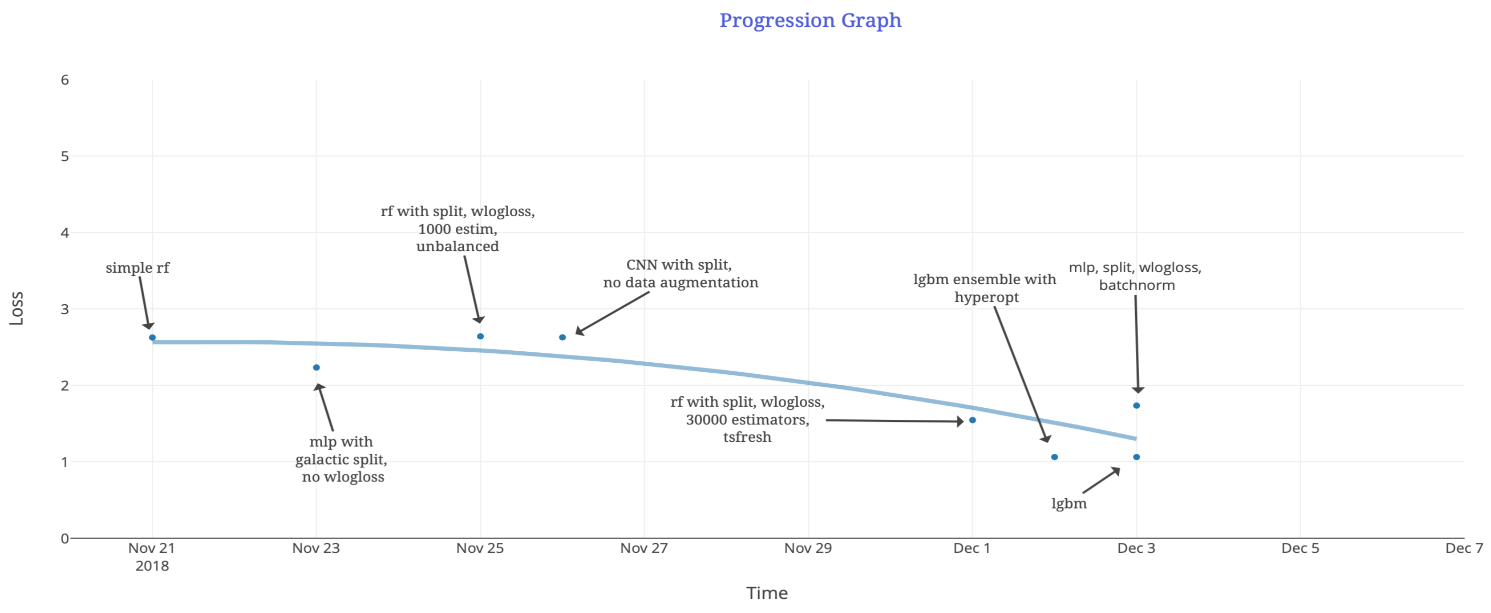


Figure 4: Progression Graph.

References

[1] Calculation of distmod, kaggle discussion. `https://www.kaggle.com/c/PLAsTiCC-2018/discussion/71871`. Accessed: 2018-12-03.

[2] Convolutional neural networks for visual recognition, stanford university cs231n. `http://cs231n.github.io`. Accessed: 2018-12-01.

[3] Cs229 machine learning, stanford university. `http://cs229.stanford.edu/syllabus.html`. Accessed: 2018-12-03.

[4] How to win kaggle competitions, kaggle. `https://www.kaggle.com/getting-started/44997`. Accessed: 2018-12-03.

[5] Strategies for flux time series preprocessing, kaggle. `https://www.kaggle.com/mithrillion/strategies-for-flux-time-series-preprocessing`. Accessed: 2018-12-01.

[6] Tsfresh, python library for time series features. `https://tsfresh.readthedocs.io/en/latest/`. Accessed: 2018-12-01.

[7] Use causality, kaggle discussion. `https://www.kaggle.com/c/PLAsTiCC-2018/discussion/69696#410538`. Accessed: 2018-12-01.

[8] What is lightgbm, how to implement it? how to fine tune the parameters? `https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-6` Accessed: 2018-12-01.

[9] Tarek Allam Jr, Anita Bahmanyar, Rahul Biswas, Mi Dai, Lluís Galbany, Renée Hložek, Emille EO Ishida, Saurabh W Jha, David O Jones, Richard Kessler, et al. The photometric lsst astronomical time-series classification challenge (plasticc): Data set. *arXiv preprint arXiv:1810.00001*, 2018.

[10] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[11] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.