Report: PLAsTiCC Astronomical Classification

Marek Topolewski

Abstract

The aim of the project was to partake in a Kaggle [1] competition called "*The Photometric LSST Astronomical Time-Series Classification Challenge*" (PLAsTiCC) organised by the LSST Project [2]. The ultimate goal of the challenge depends on classifying observation data on astronomical sources that is estimated to be generated by the *Large Synoptic Survey Telescope* (LSST). These observations are obtained by differencing images of the sky taken by the LSST. The project attempts to achieve the best galactic class predictions by employing a variety of Machine Learning techniques and data models. The report will begin by closely examining the data as well as possible methods of extracting its underlying features and approaches for augmentation of the training dataset. Subsequently, tested models with different parameter configurations, on raw and enhanced data (with feature extraction and/or augmentation methods), will be presented and evaluated. Finally, the progress will be illustrated using a progression graph representing the scores achieved by all Machine Learning models, and the outcomes of the project will be discussed.

1 Data Exploration

The dataset of the challenge is separated into *training* and *test* sets. The first includes a smaller subset of information (below 8 thousand objects) suitable for model development and testing, and the latter consists of a much larger set of observations of approximately 3.5 million astronomical sources. Each collection is divided into *observation sets* and *metadata* files (see Figure 1). Metadata provides static information in a single row for each object. Attributes in these sets include: unique object identifier, sky coordinates, galactic coordinates, flag indicating whether detected by the DDF, source redshift, host galaxy redshift, host galaxy redshift uncertainty, distance to source, extinction of light MW E(B-V), and astronomical class of the object (only in the training set). In contrast, the latter files include *time series*, also known as *light curves*, the represent object's *flux* (brightness) in six *passbands* (astronomical filters for different photon wavelengths). Each light curve has the following features: object identifier, date of the sample (in Modified Julian Data), passband, flux, flux error and a flag denoting if the source was bright enough compared the host galaxy to be detected.

	object_id	mjd	passband	flux	flux_err	detected
897896	37192802	60268.0986	2	4.364760	3.739385	0
1023013	60236596	60524.0817	0	8.955895	15.409293	0
410519	207283	59588.2190	2	35.976727	1.165331	1
804593	20228090	60462.1092	2	18.260517	17.189856	0
1033616	62050723	59944.1910	3	-16.349285	7.385664	0

	object_id	ra	decl	gal_l	gal_b	ddf	hostgal_specz	hostgal_photoz	hostgal_photoz_err	distmod	mwebv
4507	54923727	8.340517	-46.571846	312.147084	-70.231645	0	0.3141	0.3237	0.0207	41.1465	0.011
784	129490	34.453125	-5.229529	169.987075	-59.956185	1	0.3803	0.4821	0.2384	42.1664	0.019
7831	130552230	54.316406	-18.524391	209.170030	-51.015495	0	0.1226	0.1440	0.0172	39.1698	0.061
4286	49789340	4.750000	-56.637199	311.837761	-59.926647	0	0.1591	0.1666	0.0168	39.5157	0.009
1884	305891	54.667969	-27.615883	223.610785	-53.050840	1	0.2058	0.2604	0.0280	40.6031	0.009

Figure 1: Raw time series data (left) and source metadata set (right) samples. Files in "csv" format were loaded into pandas [5] DataFrame objects to allow parsing and diplayed using Jupyter notebook [6].

In the challenge's description [3] it is stated that all classes (except for 99) can be divided into two subsets: galactic and extragalactic. An object belongs to one of the galactic classes if it is located in the Milky Way, otherwise, it is extragalactic. It is also given that the redshift of the host galaxy is zero for the Milky Way, and the redshift of all other galaxies is non-zero.

As presented during the lecture providing astronomical background for the competition [16], only galactic sources are periodic, i.e. the source continuously completes cycles of flux revolutions within a specific period (see Figure 3). This property is not characteristic for extragalactic sources which are aperiodic and resemble bursts or other non-recurring events (see Figure 2).

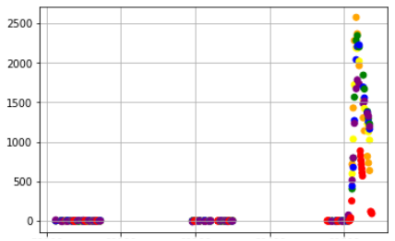

Figure 2: Flux observations in all passbands of object 7033. This source belongs to an extragalactic class 42, which indicates aperiodicity illustrated by a burst in the final observation period.

Another important property of the data is gap periods. These can be divided into *small* gaps between minutes to weeks resulting from the LSST scanning another region of the sky and large gaps exceeding six months when the source cannot be observed by LSST [3].

An interesting challenge poses the classification of the previously unseen astronomical classes denoted by number 99. Several approaches to determine the most efficient method of detecting sources of this class and obtaining corresponding probability were tested and are described in this report (see section 5).

2 Feature Extraction

Classification of time series is more demanding than a regular feature space because consecutive samples of a single object are highly correlated, hence, dedicated techniques must be applied to model their behaviour. The first possibility is to use ML models specially designed for handling this type of data, such as Recurrent Neural Network (RNN) or a Long Short Term Memory (LSMT) network [7]. This section focuses on another approach, which attempts to convert the raw data into representative attributes that are suitable for standard supervised learning. That process is called *feature extraction*, and this section will present and evaluate methods for it adopted in this project.

Host galaxy redshift allows distinguishing between galactic from extragalactic objects, which in turn have different flux revolutions (see section 1). This distinction further enables building separate models for both types and derive features that are characteristic for one type but not obtainable in the other.

The first such feature is the *absolute magnitude* which is a measure of the brightness of an extragalactic object on an astronomical logarithmic magnitude scale [8]. The attribute can be calculated using the following set of formulas:

$$f_{true} = max(\boldsymbol{f_x}) - median(\boldsymbol{f})$$
$$m = \begin{cases} -2.5 * \log(f_{true}) & f_{true} \geq 1 \\ 1 & f_{true} < 1 \end{cases}$$
$$m_{abs} = m - distmod$$

where m is the magnitude of vector $\boldsymbol{f_x}$ of flux observations for a given source with a *detected* flag set (see section 1), $\boldsymbol{f}$ is a vector of all flux values, and $distmod$ is the distance to that source (distance modulus).

In contrast, a feature that can be calculated only for galactic objects is the *periodicity* (see section 1). Multiple approaches to estimate the period of a given object has been tested: *Discrete Fourier Transform* (DDF) [9], *Lombscargle* [10] and *LombScargleMultibandFast* (LSMF) [11]. All these methods select a set of parameters of an underlying function (e.g. sinusoidal for DDF) that approximate the flux function. The *LombScargleMultibandFast* fits the model using all passbands instead of combining them into one signal, which allows for more accurate period optimisation. The DDF was deemed the fastest to computes but LSMF proved the most effective in terms of average log loss calculated using k-fold cross-validation on training data.

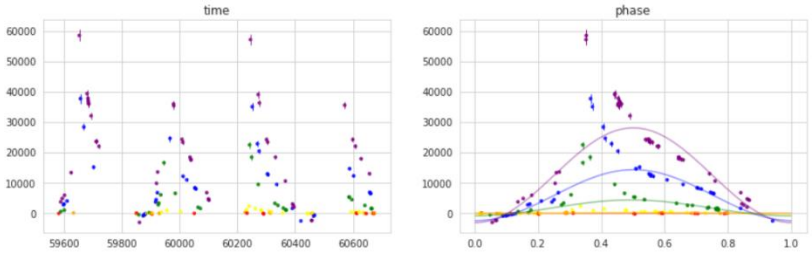

Figure 3: Time series of different passbands for a single object illustrated on a graph [4]. The left figure represents raw light curves plotted against time, whereas the right one displays the flux within a single potetnial period of that object (note that not all sources in the competition are periodic) with a fitted line illustrating the estimated underlying function generating each passband.

A feature that can be extracted for both galactic and extragalactic objects is the measure of *cross-similarity*. The attribute can be obtained using Dynamic Time Warping (DTW) [12], which produces a real value inversely proportional to the similarity of two flux functions. The method aligns two sequences of observations and it is invariant to many non-linear transformations in the time domain, which leads to better robustness. However, it is computationally expensive to compute the measure between every test object and representative flux vectors of each class from the training set and selecting a representative object for every class does not guarantee the inclusion of all training objects. Considering the above disadvantages posed by this feature, after testing it was not adopted in the project.

Finally, a vector of flux can be converted into a set of *aggregate features*. A complete and optimal solution requires a set of attributes (excluding the key) such that it uniquely identifies each class. Although this is rarely feasible, models can attempt to achieve this by selecting features maximising *information gain*, which is a measure of how discriminative a given attribute is [13]. A number of attributes have been proposed in Kaggle kernel [14], from which 26 features were selected based on highest log loss scores on training data using k-fold CV and computational cost.

3 Data Augmentation

The size of the training set is significantly smaller compared to the test set, therefore, to produce more accurate predictions the models can be fitted in a larger sample space. The process of expanding a data set is known as *data augmentation* and is achieved by generating new observations and targets based on the existing ones. This decreases model's variance error, hence, better generalisation and performance on unseen data.

The method adopted in this project includes creating new samples by adding noise to the flux values in the observations. The distribution of noise is Gaussian with the mean centred on the observed flux value for that date and variance proportional to its corresponding flux error, which ensures more accurate samples. Because the actual dates are not used in any model, the starting date remains the same for the ease of processing. The generated time series are added with an unseen object identifier and a corresponding entry in metadata which is a copy of the origin object's record.

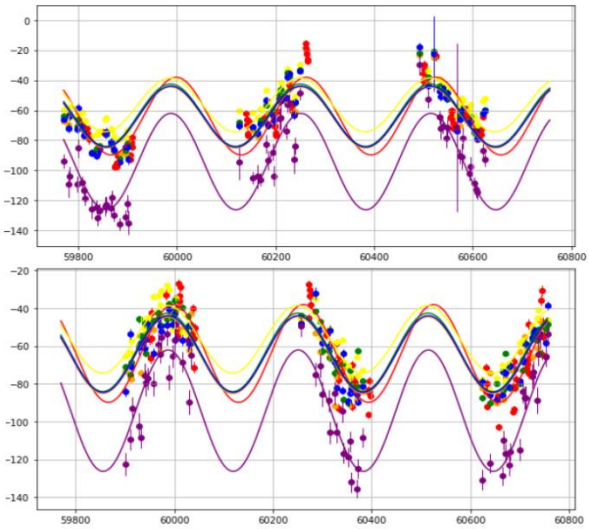

Figure 4: New sample generated based of object 4088 (class 88). The top figure represents the orignial flux plot in time, and the bottom graph illustrates a new object obtained through data augmentation with time shift of 100 units.

To further improve the robustness of the trained model, different gap periods can be simulated for galactic sources. Because *LombScargleMultibandFast* estimates the period by optimising a model to approximate underlying flux generation process (see Figure 3), the model itself can be sampled from to simulate the observations of the LSST. Observations are handled for each passband separately and "shifted" by half of the objects period (see Figure 4). The LSMF model predicts flux for these dates and Gaussian noise is added with the predicted value as the mean and variance as mean difference between predicted and true flux (for given passband and original dates). Detected flag is set with the same probability as the ratio in the original time series, and the flux error is predicted using *RandomForestRegressor* [15] (one per passband) with the following features: flux, passband, detected, noise and phase (date modulo period).

4 Raw Data Classification

To provide a benchmark for further development of ML models, classifiers with default parameters and minimal feature engineering were tested at first, and were further improved by tuning the selected models. This section describes machine learning method and models trained on raw data, i.e. raw time series or in this case simple aggregate functions such as the mean of a given attribute. Sections 4 to 6 provide an overview of the adopted ML techniques and their k-fold CV log loss score for which corresponding weighted multiclass losses on the test set are presented in section 7.

The initial naïve approach was to produce a *random-guessing* algorithm that uniformly distributes the probability of belonging to all classes. It has been also deemed profitable to include class 99 instead of assuming probability zero due to a higher score on the test set (see section 8). Although theoretically the method is bound to perform poorly, it has been determined to outperform many far more complex techniques and scoring 2.639 on the training set. Moreover, the approach was also significantly faster to compute than other methods considered in this project.

The first Machine Learning model used to classify the time series was a *RandomForestClassifier* [17]. As mentioned above, in this part of the project the focus was on tuning the selected models to improve efficiency on classifying raw data, hence, the model was fitted to the mean values of the observation set and did not include any metadata or feature extraction methods. The resulting classifier was further improved from 3.631 to 0.925 (5-fold CV log loss) by employing hyper-parameter tuning with *RandomisedSearchCV* [18]. The method (as described in [19]) sets prior distribution over the model's parameters and samples from resulting parameter space to converge to the optimal configuration.

The other approach undertaken for raw data classification was incorporated a basic neural network *MLPClassifier*, which is a Multi-layer Perceptron implemented in sklearn [20] and aim to optimise the log-loss score using a specified solver (e.g. Stochastic GD or Adam). Similarly to RFC, the MLP was first tested without tuning, which resulted in low performance as expected, because MLPs are highly sensitive feature scaling and to the initial configuration (especially the number of hidden layers) both of which are not included in this stage. Subsequently, the MLP was tuned using GridSearchCV [21], which improved the log loss (average k-fold CV) from 22.788 to 1.635.

5 Enhanced Data Classification

As described in section 3, and expanded upon in section 4, classification on raw time series does not result in informative predictions but tuning the models can result in significant performance improvements. In this section, the endeavour to construct more expert models by adopting feature extraction and data augmentation techniques outlined in sections 2 and 3 is described. To provide a comparison measure, the same underlying ML models as in section 4 will be employed.

Due to large data sets, the features were precomputed and exported into new CSV files, which can in turn be loaded into a selected Python script. In an attempt to decrease the computational cost, a *RandomForestRegressor* was trained to estimate the features, however, the performance was deemed to be insufficient to adopt it. For the same reason, the models were evaluated by CV on the training set prior to generating any predictions for the test set. Also features that were deemed uninformative in [14] were dropped to reduce model complexity (see Figure 5).

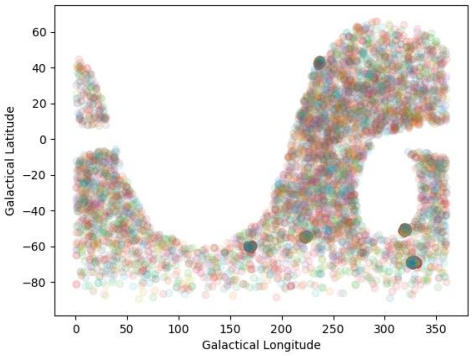

Figure 5: Class distribution plotted by galactic coordinates, where darker areas indicate higer density. The graph confirms that using these attributes will not improve model's performance.

Another challenge addressed at this stage of the project was the *unknown class 99*. The naïve approach used in raw series classification was to always assign it a constant probability, either zero or equal to all other classes (uniform). Another way would be to train an outlier detection model such as *OneClassSVM* available in the sklearn package. This approach was tested, however, accurate identification of an outlying training class sample within a set of one training class was not achieved, hence the method was not included in developed models.

The first implemented method was a *selective uniform distribution*, which is based on the random guessing approach from section 4 but uses the trivial distinction between galactic and extragalactic classes. A probability of zero is assigned to all classes that are in a different source type, e.g. for non-zero host galaxy redshift of the sample set its probability of belonging to any galactic class to zero. The probability of belonging to the remaining classes, which regardless include class 99, is uniformly distributed amongst them. Not only is the least computationally expensive method developed in this project, but it also outperforms many of them scoring 2.023 in terms of log loss on the training data.

Secondly, full feature extraction and data augmentation were applied to improve the RFC approach. All objects in the original and augmented metadata files were divided into two subsets of galactic and extragalactic sources. Appropriate features were then extracted and scaled. It had been deemed the most optimal to normalise all the extracted training features with the exception of the period and absolute magnitude, as this configuration yielded the highest k-fold score. Subsequently, two RFC models were tuned like in section 4 to predict within each subset with the highest accuracy. The results were merged my filling missing prediction classes for the other subset with zeros and exported to a single file. The final k-fold CV score of this model was 0.257 for galactic, 0.965 for extragalactic, and mean of both 0.611.

The final ML model discussed in this section is the Multilayer Perceptron train on enhanced data. Similarly to the enhanced RFC, the data was augmented, then applicable features were extracted and samples were separated into two galactic groups. Two MLPs were tuned using *GridSearchCV* to minimise log loss of predicting their corresponding classes on normalised training data. The final k-fold CV score of this model exceeded the performance of the enhanced RFC by scoring 0.326 for galactic, 0.682 for extragalactic, and mean of both 0.504. This makes it the best performing model (on training data), especially considering the asymmetric distribution of the classes, i.e. more extragalactic objects are sampled (over twice as many, see Figure 6), hence, models classifying this group well are favoured.

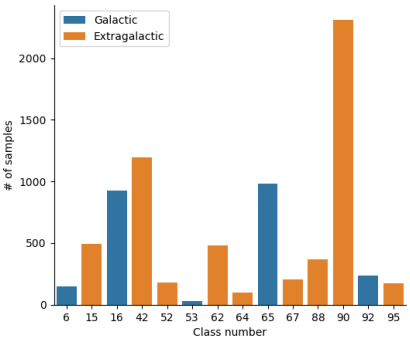

Figure 6: Distribution of objects per class in the training set.

6 Convolutional Neural Network

The final model constructed in this project is a *Convolutional Neural Network* (CNN) imported from keras [22]. The implementation is based on a Kaggle kernel describing how a CNN using raw data can be built [23]. The project further expands the kernel as it provides additional training samples by adopting data augmentation (see section 3) and converting the keras 2.2 source code to keras 1.2.2 for compatibility with the installed version of the package on the DCS lab machines.

Unlike in sections 5 and 6, in this part of the project no feature extraction methods will be employed. The time series in training data and additional samples obtained through data augmentation (see section 3) will be directly inputted into the neural network for classification. The only transformations were feature data frame reshaping and passband alignment required for the correct functionality of the Convolutional Neural Network. The CNN has a critical advantage over all previously mentioned models as it allows to preserve the spatial structure of its input. This implies that despite the lack of feature extraction the model can perform well, as it can classify the sample using the entirety of flux observations instead of its aggregations or derivative features.

To construct a neural network all observations of every sample must be converted in a form of a two-dimensional array. On one axis of that array there are passbands (zero to six), and on the other a sequence of features (flux, flux error, detected flag) of every time series in that passband of that object. Once the inputs are transformed, a target map of the true predictions is constructed where for each sample the result is encoded into a binary array (bit 1 only for the row representing the class the given object belongs to).

The neural network itself is constructed by sequentially adding different layers to the model, functional API of keras is used to propagate the intermediate results across the network [22]. Each layer is a tensor data structure and also returns a tensor, which is used as the input to the next layer. The initial layer is the input layer, which specifies the shape of the series. This layer is followed by a sequence of convolutional, batch normalisation, activation and max pooling layer. The batch normalisation layer normalises (analogous to *StandardScaler*) activations of preceding convolutional layer, which enables to use higher learning rate and reduces variance propagation [24]. The max pooling layer permits to reduce the size of the feature space (down-sampling), which improves generalisation by preventing overfitting. Above sequence of layers is repeated once more (with a reduced filter size of in the second convolutional layer) before the outputs are aggregated using tensor's mean (in a lambda layer) and finally converted into a probability distribution by employing the *softmax* as the final activation function.

Prior to appending augmented data to the original training set the k-fold CV log loss averaged 2.443. After adding the additional samples, the score improved to a mean of 1.952. On the other hand, the training time increased significantly due to the required computational cost.

7 Progression Graph

To provide a clear comparison between all methods, progression graphs were produces based on the results obtained throughout project duration (see Figure 7). Each plot consists of submission numbers on the X-axis and corresponding score on the Y-axis. The submission number uniquely identifies each submission; however, it does not reflect the date of the given attempt, yet, the ordering of remains the same. The four models used are colour-coded (see the legend of Figure 7) and shapes of the points are based on what kind of data: raw, with feature extraction, with data augmentation or feature extraction and data augmentation. Not all predictions measurements were included in this section, most notably submissions that resulted in worse performance for a given model and data type were excluded.

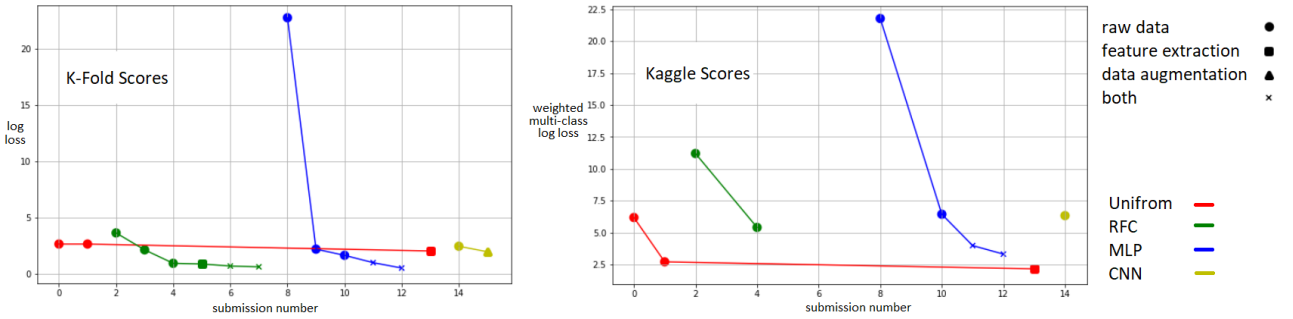

Figure 7: Progression graph generated for this project using Jupyter Notebook. Left-hand-side plot represents the K-Fold scores and the right-hand-side figure illustrates the Kaggle scores. For more details see section 7.

The left figure plots the results obtained by computing an average of 5-fold CV log loss on training data. This figure contains more observations, as it is less computationally expensive to calculate its values than generating predictions for the entire test set and submitting for evaluation to Kaggle.

The right figure consists of fewer points as a result of computational expense. However, these values reflect the true score of each model on the test set, hence, are more representative of the actual performance. Moreover, the test set also includes class 99, to which a method classification was not successfully developed in this project (see section 5).

It can be observed from the progression graphs that all models were improved as further data enhancement techniques were added. Another factor that positively influenced the efficiency, most notably in the Multilayer Perceptron, was hyper-parameter tuning.

On the other hand, the difference between the two measures appears when considering best-performing model. When examining K-Fold results the most optimal implementation the MLP with data augmentation and feature extraction. In contrast, the weighted multi-class logarithmic loss calculated by Kaggle favours selective uniform distribution (see Figure 8). Amongst others, this divergence can be a result of overfitting or lack of informed predictions about class 99. One can exclude overfitting as the root cause, due to the use of cross-validation, which prevents it and promotes generalisation.

submission	model	feat_ext	data_aug	s_kaggle	s_kfold
0	Uniform	0	0	6.182	2.639
1	Uniform	0	0	2.708	2.639
2	RFC	0	0	11.205	3.631
3	RFC	0	0	NaN	2.122
4	RFC	0	0	5.406	0.925
5	RFC	1	0	NaN	0.866
6	RFC	1	1	NaN	0.681
7	RFC	1	1	NaN	0.611
8	MLP	0	0	21.792	22.788
9	MLP	0	0	NaN	2.183
10	MLP	0	0	6.441	1.635
11	MLP	1	1	3.982	0.983
12	MLP	1	1	3.313	0.504
13	Uniform	1	0	2.158	2.023
14	CNN	0	0	6.332	2.443
15	CNN	0	1	NaN	1.952

Figure 8: Table of scores from which the progression graph was created.

8 Conclusion

The project researched and implemented many techniques in an attempt to solve the classification of astronomical time series in the PLAsTiCC challenge. The best subset of these methods was then used to train a variety of fundamentally different models and designed classifiers were tuned to obtain the highest scores.

In the scope of this assignment, the combinations of implemented models were not considered. Such a solution could prove very efficient, for example using MLP to classify extragalactic classes and RFC for the galactic subset (see section 5). Moreover, ML models specifically designed to handle time series such as Long Short Term Memory networks were not explored. If further time and resource were provided, these areas would be examined in the pursuit of finding the best-performing model and successfully help the LSST team solve the astronomical source classification problem.

Kaggle username and display name: **CS342u1633084**
Best score: **2.158** (rank 755)

References

[1] Kaggle, "*About Kaggle*", 2018. [Online]. https://www.kaggle.com/kaggle/

[2] Kaggle, "*PLAsTiCC Astronomical Classification*", 2018. [Online]. https://www.kaggle.com/c/PLAsTiCC-2018

[3] PLAsTiCC Team, "*The Photometric LSST Astronomical Time-series Classification Challenge (PLAsTiCC): Data set*", 2018. [Online]. https://arxiv.org/abs/1810.00001

[4] Mithrillion, "*All Classes Light Curve Characteristics (updated)*", Kaggle. [Online]. https://www.kaggle.com/mithrillion/all-classes-light-curve-characteristics-updated

[5] Pandas, "*Pandas Documentation*", 2018. [Online]. https://pandas.pydata.org/pandas-docs/stable/

[6] Jupyter, "*Jupyter Notebook*", Jupyter, 2018. [Online]. http://jupyter.org/

[7] Zygmunt Z., "*Classifying Time Series Using Feature Extraction*", FastML, 2018. [Online]. http://fastml.com/classifying-time-series-using-feature-extraction/

[8] Wikipedia, "*Absolute Magnitude*", 2018. [Online]. https://en.wikipedia.org/wiki/Absolute_magnitude

[9] SciPy.org, "*Discrete Fourier transforms (scipy.fftpack)*", 2018. [Online]. https://docs.scipy.org/doc/scipy/reference/fftpack.html

[10] SciPy.org, "*Lombscargle: scipy.signal.lombscargle*", 2018. [Online]. https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lombscargle.html

[11] AstroML, "*Multiband Lomb-Scargle Periodogram*", 2018. [Online]. https://www.astroml.org/gatspy/periodic/lomb_scargle_multiband.html

[12] PyPi.org, "*Dynamic Time Warping (DTW)*", 2017. [Online]. https://pypi.org/project/fastdtw/

[13] Damoulas T., "*Lecture 6: Decision Tree Learning*", University of Warwick: CS342 Machine Learning, 2018. [Online]. https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs342/cs342-lec6.pdf

[14] Iprapas, "*Ideas from kernels and discussion (LB-1.135)*", Kaggle, 2018. [Online]. https://www.kaggle.com/iprapas/ideas-from-kernels-and-discussion-lb-1-135

[15] Sklearn, "*RandomForestRegressor*", 2018. [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[16] Armstrong D., "*Astronomy and Machine Learning: PLAsTICC*", University of Warwick: CS342 Machine Learning, 2018. [Online]. https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs342/plasticc_compress.pptx

[17] Sklearn, "*RandomForestClassifier*", 2018. [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[18] Sklearn, "*RandomisedSearchCV*", 2018. [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

[19] Koehrsen W., "*Hyperparameter Tuning the Random Forest in Python*", Towards Data Science, 2018. [Online]. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

[20] Sklearn, "*MLPClassifier*", 2018. [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[21] Sklearn, "*GridSearchCV*", 2018. [Online]. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[22] Keras, "*Keras Documnetation: Convolutional Neural Network*", 2018. [Online]. https://keras.io/layers/convolutional/

[23] Higepon, "*Keras + CNN: Use time series as is*", Kaggle, 2018. [Online]. https://www.kaggle.com/higepon/updated-keras-cnn-use-time-series-data-as-is

[24] Doukkali F., "*Batch normalization in Neural Networks*", Towards Data Science, 2017. [Online]. https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c