Abstract

The goal of the challenge is to classify astronomical objects into 14 (known) + 1 (all of the other unknown) classes using time series data i.e. flux v.s. mjd. After observing the raw data, I extracted relevant features, "absolute energy", "energy ratio by chunks" and "time series complexity" to capture the characteristics. I then proceed to input raw time series data and the relevant features to 3 models, namely RandomForestClassifier, Multi-layer Perceptron and Convolutional Neural Network. It is shown that the MLP slightly outperformed RFC, and CNN is the worst model due to data sparsity. On the other hand, research shows that MLPs with 1 hidden layer with number neurons in between input and output layer size are often optimal in terms of computational efficiency and model performance. Also, the keeping sizes of convolutional layers small, combined with trial and error to test out the number of filters, help determine the optimal CNN for time series classification in our case.

Data Exploration, Feature Engineering, and Time Series Analysis

I started off by understanding the data being provided. In context, they are time series data i.e. brightness of the objects (represented by flux) against time (measured in MJD) and the metadata. Flux is measured with 6 different passband filters and data gaps appear since only one filter is used for each measurement period. The metadata consists of the astrophysical effects e.g. dust, redshift, cosmology etc. that effects the determination of time scales and colours. Unlike the time series data, they help decide where the object is, instead of what it is. In terms of size of the data, I will be working on a training set of 7848 objects to produce model and classify over 3mil objects.

I then proceed by plotting one example object from each of the 14 known classes, to identify characteristics from each passband. Below are some of my plots of object 2677, 43509,730, 745.

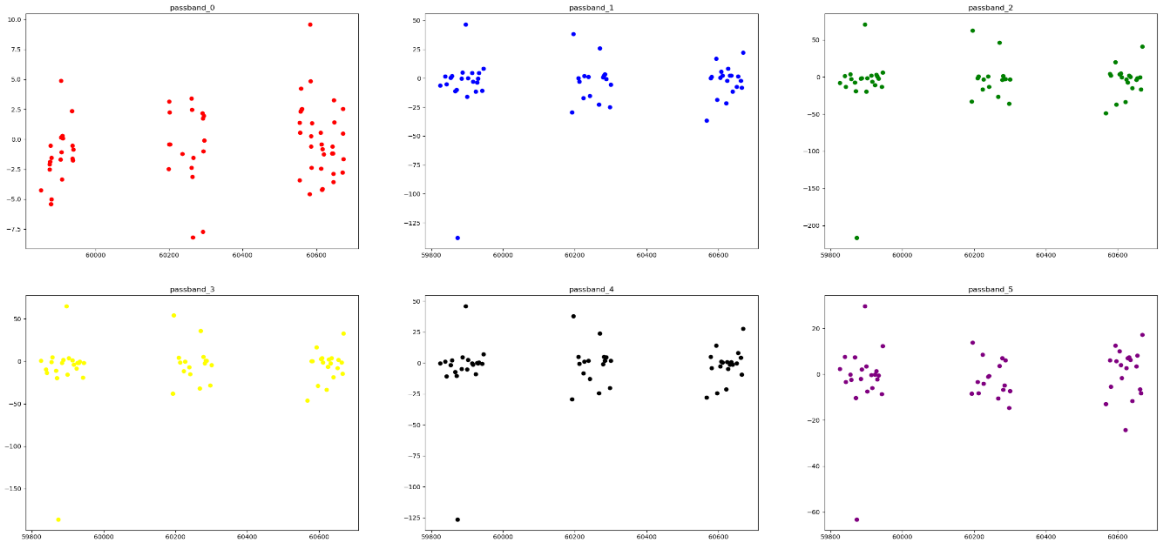


Fig. 1: object 2677 class 16 (galactic)

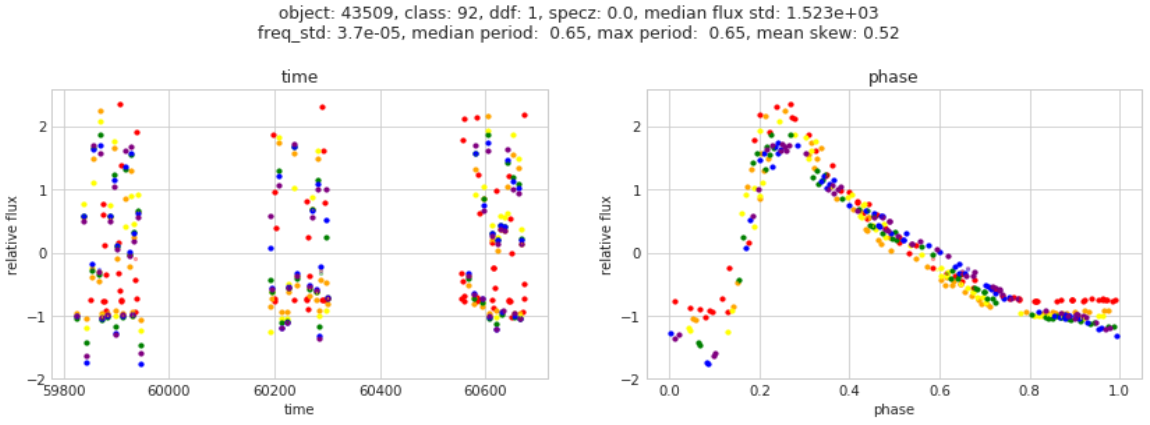


Fig. 2: object 43509 class92 (galactic) borrowed from Dr. Armstrong's slides.

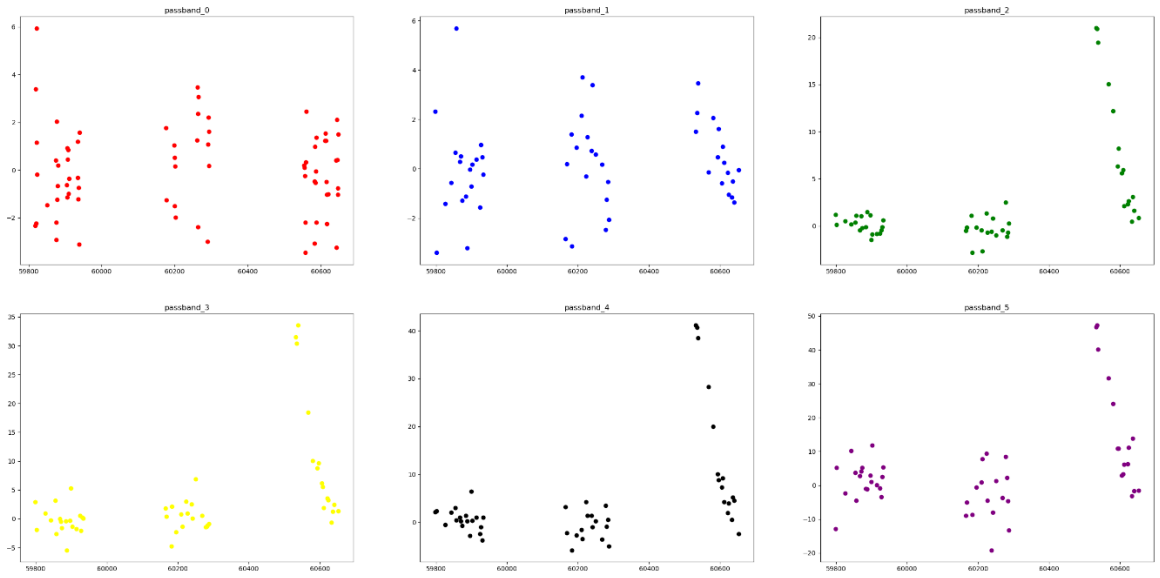


Fig. 3: object 730 class 42 (extragalactic)

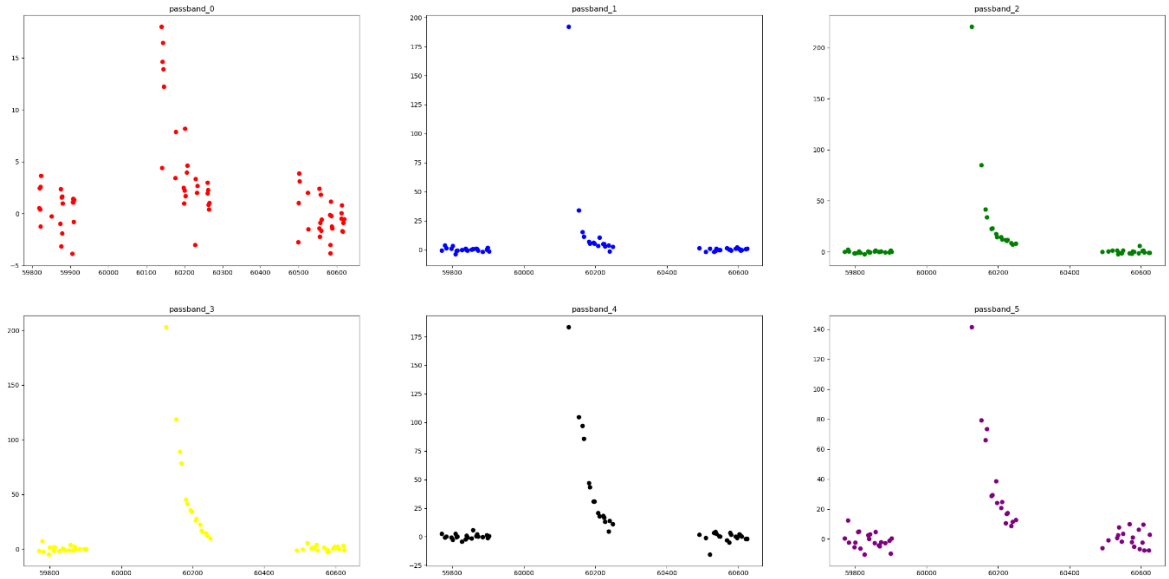


Fig. 4: object 745 class 90 (extragalactic)

Observations

1. Classes can be divided into two main types, galactic and extragalactic. For galactic objects (variable stars), e.g. object 2677 & 43509 from above, one could observe a periodic change of flux over time. Whereas for extragalactic objects (burst events), e.g. object 730 & 745, one could see a sudden increase in flux in some of the passbands.
2. When considering each passband separately, magnitude of flux of different classes varies perhaps due to the different distance away from the Earth or some of the photons being absorbed when passing through the Milky way. Either way, this gives hints on their respective locations in the Universe.
3. For the galactic objects, the length of periods, time to reach the maximum/minimum flux are all different. These are good characteristics to distinguish different galactic objects. For example, from fig.2, one could observe the time to reach the maximum flux is shorter than the time to decrease back from the maximum.
4. For the extragalactic objects, although it is often the case that only part of the burst event is captured (due to the data gaps), it is still clear that the rate of increase/decrease of flux during the event is different for different extragalactic objects.
5. Again, for the extragalactic objects, one could see that not all of the passband filters could capture the burst event. For example, one could observe the burst event of object 730 is only captured in passband 2,3,4,5. Whereas, for object 745, all the passband filters seem to capture the event distinctively in measurement period 2.

In order to engineer features to capture the observations mentioned above, I first tried to calculate the skewness of the time series. This could potentially distinguish galactic and extragalactic objects. But I realised it would be unfair to make comparisons between objects since the bursts might happen in different measurement periods. Further, even if I consider the skewness in each period, the skewness calculated would not be accurate due to the data gaps.

Firstly, I decided to work with the absolute sum of energy of the time series (abs_energy). This is the sum over the squared values of all the observed flux. I think this is suitable since, as mentioned above, within the galactic objects, the sum of squared flux in each passband could measure how bright the object is and potentially help predict how far away the object is from the Earth. Also, I suspect the galactic objects would generally give larger values since extragalactic objects are one-off burst events. Of course, one could argue that the one-off event would provide much higher "energy" of the time series, but I think this is seldom the case and this could be balanced by the other features I have created.

Secondly, I calculated the energy ratio by chunks (energy_ratio_by_chunks). This calculates the sum of squares of chunk i out of 3 chunks expressed as a ratio with the sum of squares over the whole series. I picked 3 chunks here due to the data gaps. This feature starts to compare the flux across the 3 periods. For extragalactic objects, the particular measurement period of the burst event will have a larger value. When comparing the galactic objects, this feature helps show how fast the object is moving away from the Earth (as object moves further away, the flux decreases. Faster the movement, larger the decrease in the energy ratio).

Lastly, I calculated the complexity of the time series (cid_ce). From the paper, *An efficient complexity-invariant distance for time series*, by *Batista, Gustavo EAPA, et al* (2014), they introduced this measure to find the complexity i.e. a more complex time series has more peaks, valleys etc. It is essentially the Euclidean distances of all the consecutive data points. This helps identify and measure any sudden increase/decrease in flux.

For data augmentation, all of the above features are calculated for each of the 6 passbands for each object. So that the characteristics of the times series in different passband filters could be captured. Specifically, for energy ratio by chunks, I divided the data in each passband into roughly 3 periods and calculated the energy ratio of each of the period for comparison.

Machine Learning Models: Artificial Neural Networks and Deep Learning

Kaggle Displayname: CS3421605217ChristopherYau, Ranking: 875

In this section, for each of the model choice, I will briefly explain how the model works and how I tune them. Then I will describe their performance and the reasons for their pros and cons.

Random Forest Classifier (RFC)

An RFC is an estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive. The split that is picked, is the best split among a random subset of the features. Due to the randomness, the bias of the forest usually slightly increases with respect to the bias of a single non-random tree. However, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better model. Further, I used GridSearchCV, which is an algorithm to experiment all combinations of the possible parameters given as input with respect to the training data, to find the best combination of hyperparameters through cross validation. I chose to tune the number of estimators, minimum number of samples required to split an internal node, number of features to consider for best split and the maximum depth of the tree. Tuning these parameters help prevent the model from overfitting to the training data.

In Task 4, I first tried to input raw time series data. But this created lots of problems due to data sparsity. The number of flux readings for each object is different, varying from less than 250 up to more than 350. In addition, readings are taken with different passband filters and the measurement periods are not the same as well. So, it is very hard for the model to make comparisons between objects and gives a bad performance of 8.672. I then experimented with

simple functions as input, such as min, max, mean and standard deviation etc. and my engineered features in Task 2. Both of the two cases improved my model accuracy to 6.101 and 6.337 respectively. I think the reason behind is RFC only works well when there is not a strong, qualitatively important relationship among the features in the sense of the data. In these 3 cases, raw time series data has a strong dependence on each consecutive data, and my experiment shows that RFC is unable to pick up features from raw data, until I input the engineered features, which are relatively more distinct, in the latter two cases.

Nonetheless, RFC is a relatively efficient algorithm in terms of training time. The prediction results provided me hints on producing better results with other models.

Multi-layer Perceptron Classifier (MLP)

An MLP is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. The power of neural networks come from the ability to learn the representation in the training data and how to best relate it to the output variable. In this sense neural networks utilise backpropagation for training and learn a mapping. Mathematically, they are capable of learning any mapping function. The predictive capability comes from the multi-layered structure of the networks. The data structure can learn to represent features at different scales and combine them into higher-order features. One of the advantages is MLP can learn non-linear models. But some disadvantages are MLP with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore, different random weight initializations can lead to different validation accuracy. Also, extra care is needed when tuning the model hyperparameters.

When considering how to choose the hyperparameters to feed into GridSearchCV, I borrowed some ideas from the book *Introduction to Neural Networks for Java* by *Jeff Heaton*. First problem is the number of hidden layers.

No. hidden layers	Result
0	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.

For the complexity of time series classification problems, I don't think linear separable functions can do classification accurately. So, I have to decide whether to use 1 or 2 layers. After reading *comp.ai.neural-nets* FAQ, I found that there is a consensus that the performance improvements with a second (or third etc.) hidden layer are very few. Considering together with computational efficiency, I decided to go with 1 hidden layer MLP. Again, from *Jeff Heaton*'s book, I borrowed some rules to test out how many neurons to use within the hidden layer.

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

Lastly, together with different number of neurons, I also tried out different regularisation values (alpha), activation function, solver, and learning rate to complete hyperparameter tuning for MLP.

In terms of performance, using the similar methodology experimenting with RFC, I tested MLP with simple functions and my engineered features. Inputting raw data into MLP gives a better performance than RFC. This may be due to the fact that MLP being able to capture relationships between consecutive data and combine them into higher-order features. When inputting my engineered features to MLP, the performance surprisingly worsen from inputting simple functions. This may be due to the choice of the engineered features and also having engineered the data might cause loss of information.

Convolutional Neural Network (CNN)

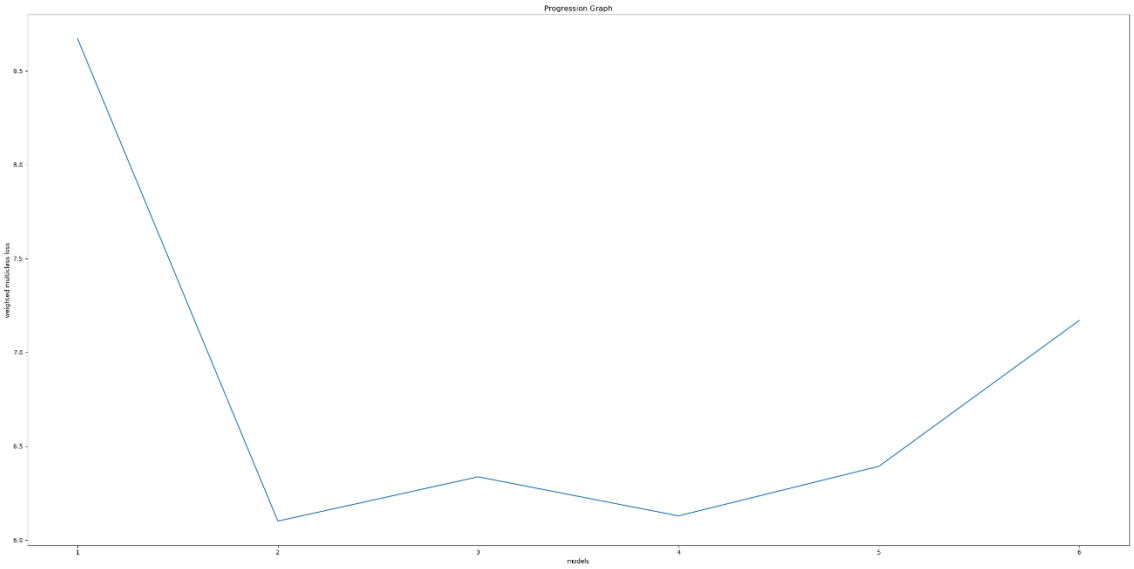
CNN is originally developed for image classification problems, where the model learns an internal representation of a two-dimensional input. But this same process to one-dimensional time series in our case. The model learns to extract features from sequences of observations and how to map the internal features to different activity types. The benefit of using CNNs for sequence classification is that they can learn from the raw time series data directly, and thus do not require expertise in astronomical knowledge to manually engineer input features. The model can learn an internal representation of the time series data and ideally achieve comparable performance to models fit with engineered features.

I produced a model with two 1D CNN layers, followed by a dropout layer for regularization, then a pooling layer. I defined CNN layers in groups of two in order to increase the chance for the model to learn features from the input data. Since CNN learn very quickly, I added a dropout layer to help slow down the learning process and hopefully result in a better model. The pooling layer reduces the learned features to 1/4 their size, leaving only the most essential elements and controlling overfitting. After the CNN and pooling, the learned features are flattened to one long vector and pass through a fully connected layer before the output layer used to make a prediction. The Adam version of stochastic gradient descent is used to optimise the network, and the categorical cross entropy loss function is used given that this is a multi-class classification problem. In particular, having read the paper *Rectified Linear Units Improve Restricted Boltzmann Machines* by *Geoffrey Hinton* allows me to understand why ReLU activation layers perform better than other nonlinear functions like tanh and sigmoid. The network with ReLU activation layers is able to train faster (due to computational efficiency) without making a significant difference to the accuracy. It also helps to solve the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers.

For parameter tuning, I tried out different values for number of filters and size of kernels. In both parameters, I calculated the average performance accuracy by running the model 10 times with the specific values and the variance of accuracy. Then I picked the one with highest accuracy and lowest variance. On the other hand, I tried to borrow some ideas from the paper *Rethinking the Inception Architecture for Computer Vision* by *Christian Szegedy, Zbigniew Wojna, et al (2015)* when considering the number of convolutional layers and max pooling layers. Since computing the activations of a single convolution is much more computationally expensive than traditional MLPs, it is better to reduce the number of convolutions. Any reduction in computational cost results in reduced number of parameters. This means that with suitable factorization, we can end up with more disentangled parameters and therefore with faster training. Having tried putting 3 convolutions and the performance accuracy did not increase as much as adding the second one, I decided to put only 2 layers. Further, from the paper, it is shown that convolutions with larger filters tend to be disproportionally expensive in terms of computation. So, when combined with the trials I ran, I decided to pick 2 convolutions with both sizes of 3.

In terms of performance, the same problem in Task 4 and Task 5 of data sparsity when extracting the raw time series data occurs. In Task 6, I picked a window of 256 time-steps (multiple of 32). This means I have to fill in 0s for the objects with less than 256 readings, and sample 256 readings randomly for objects more than 256 readings. I noticed that this methodology lost a lot of information from the raw data, which yield a score of 7.17.

Progression Graph & Discussion



1. RFC with raw time series data
2. RFC with simple functions
3. RFC with engineered features
4. MLP with raw time series data
5. MLP with engineered features
6. CNN with raw time series data

Overall, I conclude that MLP and RFC generally perform better than CNN for this time series classification challenge. Possible reasons are me not tuning the model correctly, adding more convolutions may acquire more characteristics, or the raw time series data I inputted is sparse and is not representative of the entire time series. However, there are also uncertainties in reaching this conclusion, e.g. the simple functions and feature engineering I chose to calculate etc.

There are results that I was not expecting. For example, theoretically CNN with raw time series data should be able to perform roughly the same as MLP with engineered features, but for my case, the errors are 7.17 v.s. 6.393. The underlying reasons could be the way how I input my raw time series data and the features I chose to engineer. Going forward, in terms of utilising the given datasets, I hope to gain deeper astronomical knowledge to better utilise the available data. In terms of feature engineering, I want to explore more into different feature engineering techniques on both the whole series and intervals, e.g. time warp edit distance and Derivative Transform Distance. I also want to figure out a better way to pre-process the raw time series data e.g. how to best deal with missing data at a particular time/passband. And I believe improvements in the abovementioned areas will better help me determine which model/type of data to use and produce better predictions.