MACHINE LEARNING ENGINEER NANODEGREE

Capstone Projects

OTHER EARTHS

Exoplanet-star Classifier

Darsh Kaushik

October 27th, 2020

DEFINITION

Project Overview

The question of the century perhaps, is there another blue dot of life out there in this humungous universe? Are we alone? Is there any other ball of mass which could sustain life like our Earth?

As the space technology advances day by day, the researchers hunting the potential exoplanets are bombarded with data from the satellites observing the distant solar systems from the **Kepler's second mission: K2**.

Using the **transit method** to detect brightness changes for longer periods, whenever a revolving exoplanet happens to be in the path of the satellite and the sun, the intensity of light (flux) received by the satellite from the star dims, indicating the presence of a potential exoplanet.

In this project my main focus would be to use the time-series data of flux received by the satellite for pattern recognition and identify the stars with potential exoplanets revolving around them.

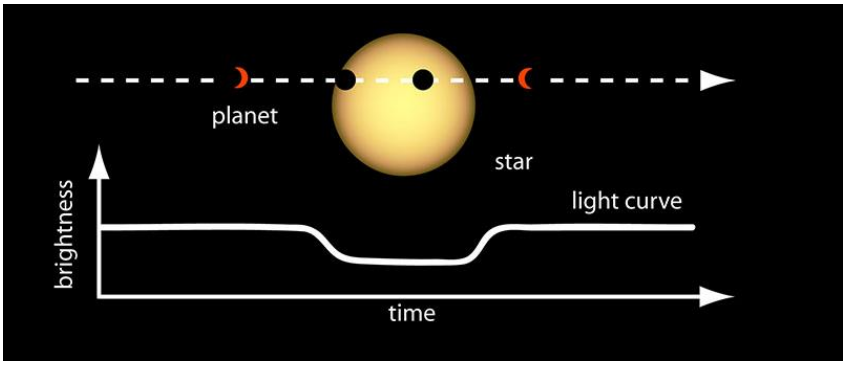


Figure: The Transit Method

Courtesy: NASA

Kepler Mission Overview

Transit method for detecting exoplanets

Problem Statement

Identifying the solar systems with potential exoplanets using the sequential data of light intensity captured in K2 mission.

The associated tasks include:

- Realizing periodic patterns in the flux data to mark the presence of revolving bodies
- Choosing an algorithm robust enough to capture the sequential, local and spatial cues present in the data.
- Smoothing out the data to remove noise and enhance the dimming in flux.
- Mitigating the effects of imbalanced classes in the dataset

Metrics

I will use **accuracy** as my evaluation metric, because this metric was used in the benchmark model. As this is a Kaggle dataset which is not heavily, it is very difficult to judge the performance on parameters which are not previously implemented.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total}}$$

ANALYSIS

Data Exploration

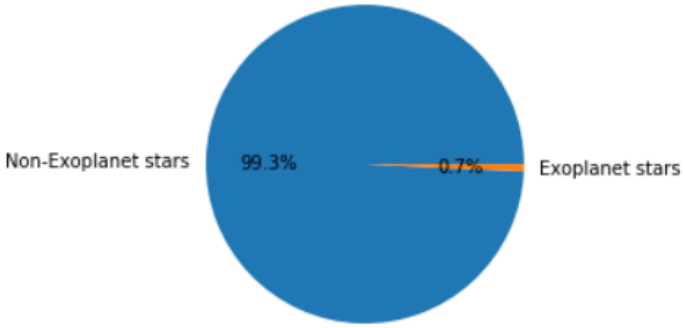
I found a Kepler labelled time series dataset named as "Exoplanet Hunting in Deep Space" on Kaggle. The datasets were prepared late-summer 2016 and over 99% of this dataset originates from Campaign 3 of the K2 mission.

The dataset has **3,197 feature columns each corresponding to a timestamp for the time-series**. For each star there are 3,197 columns with flux values over time. When plotted, we can get the curve of flux vs. time as shown in the visualizations.

The training data has 5,087 rows and 570 in the test set each containing the flux data of a star, where **each star is labelled as 2 if it is an exoplanet star and 1 if it is a non-exoplanet-star**.

Class distribution:

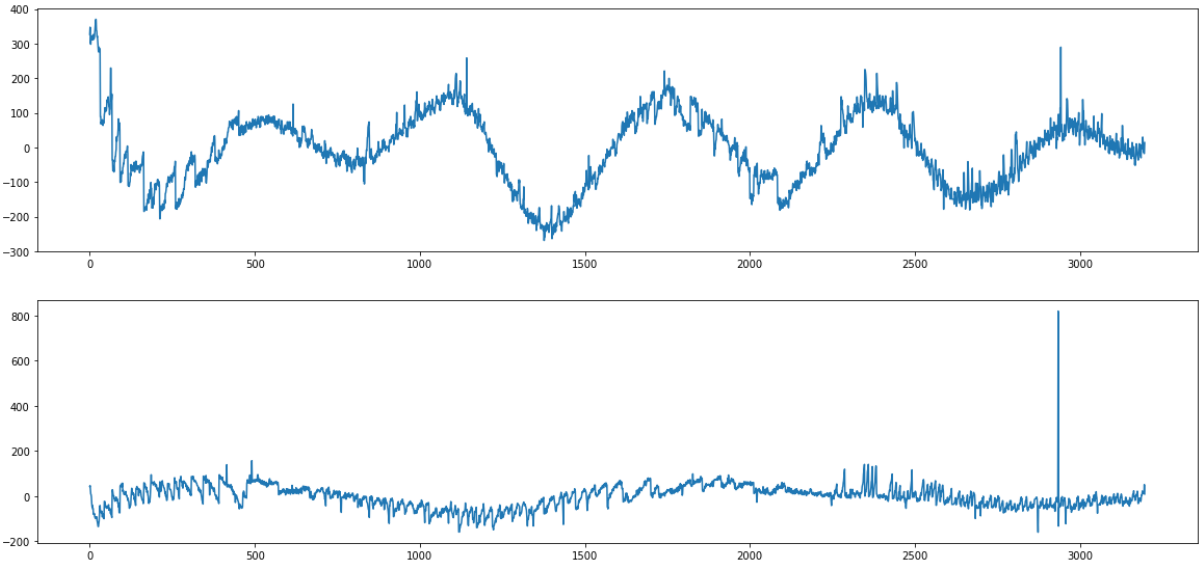
The **data is highly imbalanced**, out of 5,087 stars only 37 stars labelled as exoplanet stars in training set and only 5 out of 570 stars in the test set. Because of such skewness in data, it is very difficult to predict the exoplanet stars



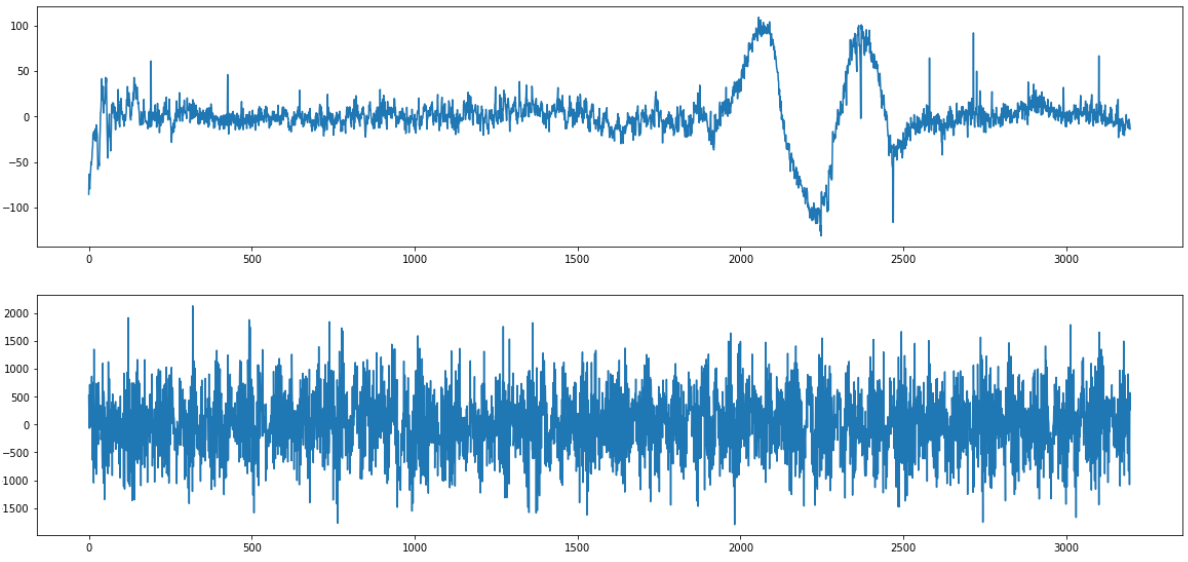
Kaggle Dataset

Data Visualization

A few Flux vs. Time graphs for exoplanet stars:



A few Flux vs. Time graphs for non-exoplanet stars:

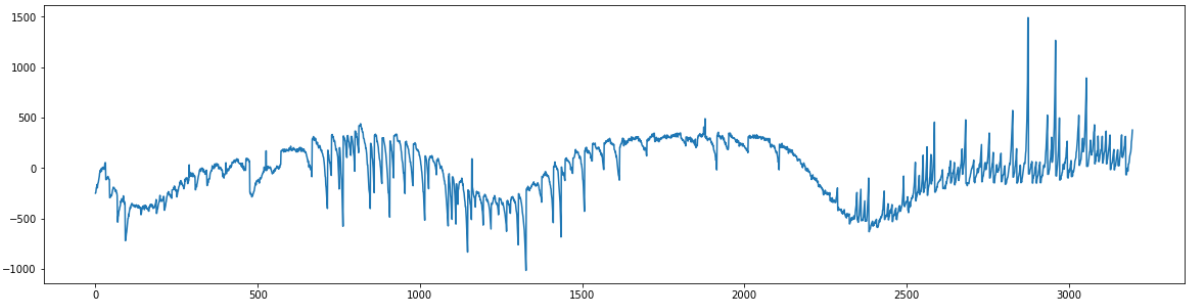


METHODOLOGY

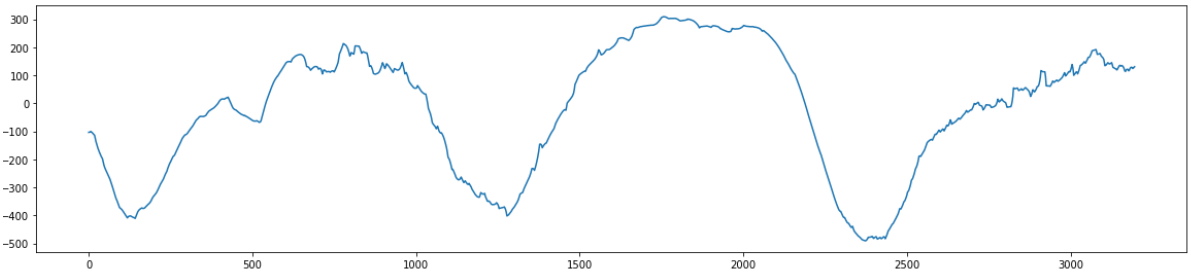
Data Preprocessing

From the above plots it is visible that the **data has high frequency noise** which might hinder the model's prediction. Therefore, the first step would be to smoothen the flux time-series and create balanced batches for training.

Plot before smoothening:



Plot after smoothening with filter of size 100 timestamps:



Also, the data cannot be used directly for making batches for training purposes because it is highly imbalanced. It is important to **oversample minority class** in each training batch to some degree in order to attain proper training results.

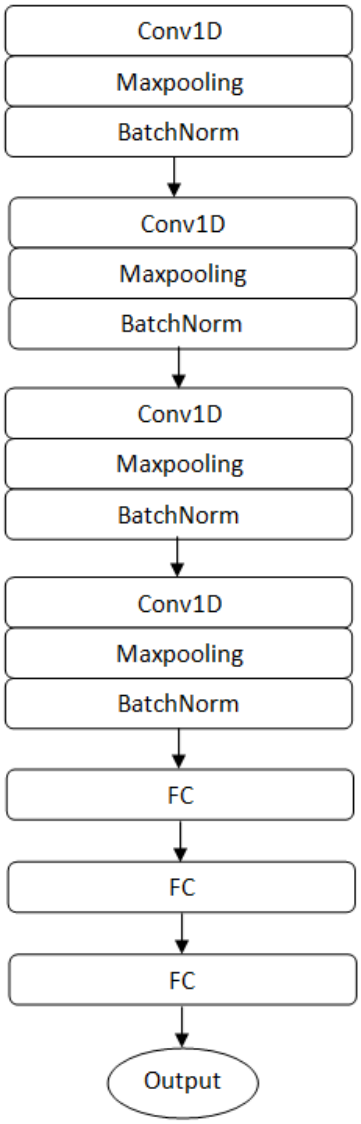
Implementation

Data Preprocessing:

I chose **1-dimensional uniform filters (uniform_filter1d) in SciPy** to average-out the time series and remove noise. The size for the filter was determined by testing some values and was fixed as 100 timestamps.

To overcome class-imbalance, I used **WeightedRandomSampler** for batch making purposes and passed these weights to the estimator while training.

Model:



Regarding the model architecture, in order to identify exoplanet stars, the **local recurring patterns** such as lowering of flux must be detected from the available flux time-series. To achieve this, I use a **combination of 1-dimensional convolutions, max-pooling and batch normalization** followed by **fully connected layers**.

1-d convolutions act like local filters for features and helps to extract the spatial aspects of the data, pooling will help in further amplifying the detected features and reduce the dimensionality to be further apply the fully connected layers on.

Loss:

Initially I tried to use a custom-made weighted binary cross entropy loss function, but ended up using **BCELoss** in pytorch as it gave better results, although I had to increase the sampling of the minority class.

Optimizer:

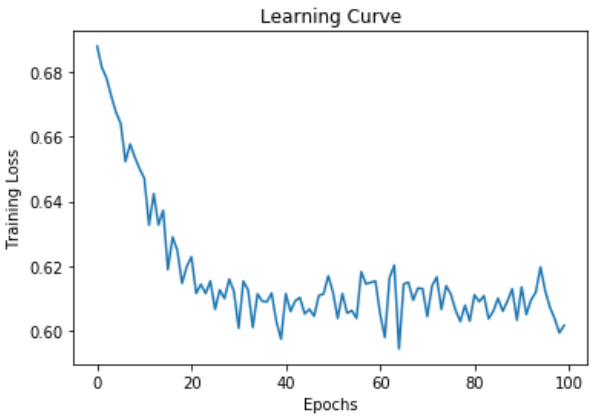
I tested both **Stochastic Gradient Descent (SGD)** and **Adam optimizer** but none of them reached convergence in even **100 epochs,** whereas the benchmark model got the results in just 40 epochs. I ended up using SGD as it performed better in comparison.

Refinement

Although there are not many notebooks available on Kaggle for this dataset, I was able to find a particular notebook (linked below) which used some of the techniques for the deep learning model mentioned above and was able to attain an accuracy of more than 90% on the test set. Therefore, I would consider the model in this notebook as my benchmark model.

The model in this notebook had only three fully-connected layers whereas mine has 4. The reason is to not to drastically change the number of nodes in each layer. Therefore, I even changed the number of nodes and gradually decreased them.

Even after 100 epochs, the model did not converge properly although it was able to get an accuracy of about 99% which is the same as the benchmark model, although it was not good at predicting the positive examples.



Benchmark model

Results

Due to very few positive examples, the loss and accuracy of training set itself was used for validation. The final architecture and hyperparameters were chosen because they performed the best among the tried combinations. For a complete description of the final model and the training process, refer to figure in page 7 along with the following list:

- The size of the filters of 1D convolutional layers was 11 timestamps.
- The first convolutional layer learns 8 filters, the second learns 16 filters, the third learns 32 filters, and the fourth learns 64 filters.
- The convolutional layers have a stride of 1.
- The max –pooling layers are of size 4 timestamps and have a stride of 2. Batch normalization is used after each max-pooling layer.
- For the convolutional and fully-connected layers I used **ReLU** as the activation function except the last fully-connected layer for which I used **Sigmoid.**
- 2 kinds of drop-out layers are used with probability 0.5 and 0.3 respectively.
- The training runs for 100 iterations.
- The classifier can reliably detect non-exoplanet stars.
- The classifier has an accuracy of 99.12%.
- The accuracy turns out to be so good because most of the stars are non-exoplanet stars and the classifier can reliably classify them.

Justification

As also discussed in my proposal, in order to classify such stars, it is important to detect the regular spatial patterns over time (which may involve lowering of the flux) to estimate whether there is any body revolving around the star. From the perspective of a deep learning model it is possible to use consecutive fully connected layers but as the feature-space is large the training will take more time as we would need larger number of layers.

Therefore, using convolutions proves to be a better a approach not just because of less number of parameters but also because the **local recurring patterns** such as lowering of flux must be detected from the available flux time-series. That is why I chose to use a **combination of 1-dimensional convolutions and pooling** followed by **fully connected layers**.

Additionally, the max-pooling helps in dimensionality reduction, making the classification easier for the following fully-connected layers.

CONCLUSION

Reflection

The process used for this project can be summarized using the following steps:

- An interesting problem and relevant data is collected.
- The data was analyzed, visualized and pre-processed.
- A benchmark model was chosen.
- The architecture was designed by multiple testing.
- The classification was trained with different ratios of over-sampling for minority classes and different optimizers until a good accuracy was obtained.
- The model was deployed as a sagemaker endpoint

The interesting transit method:

Using the **transit method** to detect brightness changes, the K2 mission entails a series of sequential observing "Campaigns" of fields distributed around the ecliptic plane and offers a photometric precision approaching that of the original Kepler mission. After looking onto a particular solar system for longer periods, whenever a revolving exoplanet happens to be in the path of the satellite and the sun, the intensity of light (flux) received by the satellite from the star dims, indicating the presence of a potential exoplanet.

Training difficulties:

Because of the skewness in the dataset, with just 37 examples of exoplanet stars, even after oversampling the model finds it very difficult to predict them.

Tools, resources, libraries and frameworks used: Python 3, Jupyter Notebook, Pytorch, Numpy, Pandas, Matplotlib, Sci-kit learn, Scipy.

Improvement

The data is not very adaptable for deep learning although convolutions tends to show good results. The class imbalance even in the test set proves to be a major hindrance in predicting the minority class.

In order to reduce the effects of instrumentation and measurement noise, it may be possible to achieve better results by using more than one channels as input data model where each channel corresponds to different sizes of uniform filters.

Another interesting approach for the problem may be sequential models like **LSTMs** as they might be able to detect the periodic long-term patterns in the flux-time series.