Exoplanet Hunting in Deep Space

Henry Gardner

CSC 240 Final Project Report, Fall 2020

University of Rochester

December 13, 2020

Abstract

The aim of this project is to predict locations of exoplanets in deep space via changes in fluxes or light intensities over time. Using training and testing data sets, various techniques and manipulations to the data showed that a relationship can be made between attributes allowing for the discovery of an exoplanet(s). The goal was to formulate the differences in accuracy using the Naïve Bayes, K Nearest Neighbors (K-NN), and 1D Convolutional Neural Network machine learning prediction methods. This would allow for comparisons to a parametric, non-parametric, and deep learning algorithm. It was determined that the Naïve Bayes approach was incredibly inaccurate due to the parametric approach which assumed independence between the data attributes. The KNN Test fixed this by approaching prediction non-parametrically, which was shown to be exponentially more accurate than the Naïve Bayes approach. Lastly, a 1D Convolutional Neural Network was proven to converge to an accuracy score of over 85% with only 40 epochs, which can improve with a higher learning rate. The limitations and process of discovering these results are discussed along with the mathematical assumptions and theory.

1 Introduction

Before the analysis begins, a brief introduction describing the importance of the data and the background information will be articulated. To begin, an exoplanet is a planet outside our solar system. They are important to discover as they offer significant inside into our own planet. Identifying exoplanets can help generate and understand how planets are formed, evolve, rotate, etc. Exoplanets themselves do not emit light, however, the stars they orbit do. Changes in flux intensity over a certain period may be evidence that there is an orbiting body (exoplanet) around a star. Usually, there is a dimming effect of the flux over a considerable amount of time. The dataset came from Kaggle[1], but the data itself was a product of Nasa's Kepler Space Telescope and is part of an ongoing effort in locating exoplanets.

2 Theory

As 3 main prediction methods were analyzed, each will be depicted in a mathematical perspective stating any assumptions used in the analysis. Firstly, a discussion of the Naïve Bayes Classifier. It is important to note that when using this method, it is assumed that all the attributes are independent. In this case, this means that the columns in the data sets (i.e. FLUX.1, FLUX.2, etc.) are independent of each other, meaning that each column's value is not directly dependent on another column's value. This is a significant assumption, especially because the location of an exoplanet is directly correlated to the change in light intensity between attributes. However,

[1] WΔ. (2017, April 12). Exoplanet Hunting in Deep Space. Retrieved December 12, 2020, from https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data

even with this assumption, it still enables prediction and allows for dimension-wise probability analysis. This classifier uses the sample mean and a diagonal sample covariance matrix for each class, which requires prediction/estimation of 2d parameters where d refers to the dimension of the of this sample covariance matrix. Using this, we can formulate the likely hood for a class c_i for dimension X_j as

$$P(x_j|c_i) \propto f(x_j|\mu_{ij}, \sigma_{ij}^2) = \frac{1}{(\sqrt{2\pi}\sigma_{ij})} \exp\left\{-\frac{(x_j - \mu_{ij})^2}{2\sigma_{ij}^2}\right\} \quad (1)$$

After using the naïve assumption that all the covariances are zero, we can rearrange this to equal

$$P(x_j|c_i) \propto f(x_j|\mu_{ij}, \sigma_{ij}^2) = \prod_{j=1}^{d} P(x_j, c_i) \quad (2)$$

In words, this means that the joint probability has been decomposed into a product of the probability along each dimension as required by the independence assumption. Training the Naïve Bayes classifier in Python is computationally very fast as it has a complexity of only O(nd).

The K-NN classifier is a little different as the approach is used for estimating non-parametric instances. This is equivalent to saying that there is no underlying assumption on the joint probability density function. This will show an increased accuracy of prediction at a more intense computational complexity. This classifier trains the data based on a point x in the dimension of the data frame and comparing that to K of its closest neighbors in the d-dimensional hyperball with radius r around that test point x. This can be seen through the following equation of hyperball closeness:

$$B_d(x, r) = \{x_i \in D |\ \|x - x_i\| \leq r\} \quad (3)$$

It is important to note that $\|x - x_i\|$ is the standard Euclidean distance between x_i and x, but other distance metrics can also be used (without loss of generality). Then using

$$K_i = \{x_j \in B_d(x, r) |\ y_i = c_i\} \quad (4)$$

to denote the number of points among the K nearest neighbor, we can see that the conditional probability density at x can be estimated as the fraction of points from c_i that lie within the hyperball divided by its volume. Therefore, the predicted class of x is:

$$\hat{y} = argmax_{c_i}\{P(c_i|x)\} = argmax_{c_i}\left\{\frac{K_i}{K}\right\} = argmax_{c_i}\{K\} \quad (5)$$

Thus, because K is fixed, the K-NN classifier predicts the class of x as the majority class among its K nearest neighbors. Clearly, this will be much more accurate as the non-parametric analysis allows for more prediction based on the immediate assumption that the attributes can be dependent. This leads to a computational complexity of O(nd+kn) or O(ndk) for assumed fixed K.

Lastly, a neural network (inspired by a biological neuronal network) acts as a processing unit comprised of abstract neurons in the form of a weighted graph. They aggregate incoming signals and then apply a function to those signals to generate an output. Each artificial neuron has a set of inputs, weights, and bias values. MLP (multi-layer perceptron) is a neural network with distinct layers of neurons. Each intermediate layer is a hidden layer between the input and output layers. For this analysis a 1D Convolutional Neural Network was used which means that the data which was input through these artificial neurons was in the form of a vector, or one axis. A Convolutional Neural Network (CNN) connects adjacent subsets of neurons in a layer L into a single neuron in the next layer L+1. The general idea behind 1D CNN is as follows: consider x = $(x_1, x_2, x_3, \dots, x_n)^T$ to be the input vector with n points. By assumption a CNN assumes that the input points are not independent, this is very useful for this analysis, as we saw from the Naïve Bayes limitation. We can define another vector w = $(w_1, w_2, w_3, \ldots, w_k)^T$ for $k \leq n$ (k is referred to as the window size) to be a vector of weights called a 1D filter. It is important to note that the weight value is a parameter that transforms the input and the result determines whether that output is observed or passed to the next layer in the network[2]. Using this we can denote $x_k(i) = (x_i, x_{i+1}, x_{i+2}, \ldots, x_{i+k-1})^T$ to be the window of x of length k starting at position i, where $1 \leq i \leq n - k + 1$. A 1D convolution between x and w (denoted using the symbol *) can be defined as:

$$x * w = [sum(x_k(1) \odot \mathrm{w}) \ldots sum(x_k(n-k+1) \odot \mathrm{w})]^T \tag{6}$$

Where $\odot$ denotes the element-wise product or Hadamard product such that:

$$sum(x_k(i) \odot \mathrm{w}) = \sum_{j=1}^{k} x_{i+j-1} w_j \tag{7}$$

This happens for i = 1, 2, … , n-k+1, therefore resulting in a convolution value of vector length n-k+1. Since the analysis used a deep CNN, there were multiple layers of these convolutions that connected in hidden layers eventually resulting in the final output layer. This output can then be accessed based on its accuracy of prediction. An epoch is defined to be an arbitrary cutoff deemed as "one pass over the entire dataset." This is important as it separates the training into phases, which is very useful for logging and evaluating accuracy. Different epoch values were used when training the CNN to test convergence at different learning rates.

Python libraries did the heavy lifting for these mathematical procedures, so the code was somewhat easy to implement.

3 Python Libraries and APIs

The analysis required several different machine learning methods and techniques that have already been developed by other programmers. Using their foundations, generating the classification and regression models was straight forward as the library syntax was clear and basic to implement. For both the Naïve Bayes and KNN Test approaches, the sklearn library was used which allowed for several free implementations of classification, regression, and clustering algorithms. After preprocessing, each technique required only a handful of lines of code. For the 1D Convolutional Neural Network, the Keras library (product of the TensorFlow library) was the optimal path. The Keras library has a Sequential Model that allows for a plain stack of layers consisting of one input and one output, making it the perfect library for the deep learning analysis. In addition, the seaborn and matplotlib libraries were used to graph the convergence of the 1D CNN over each iteration and to perform basic graph analyses used throughout the preprocessing. Other libraries were used and referenced in the remaining portion of the report.

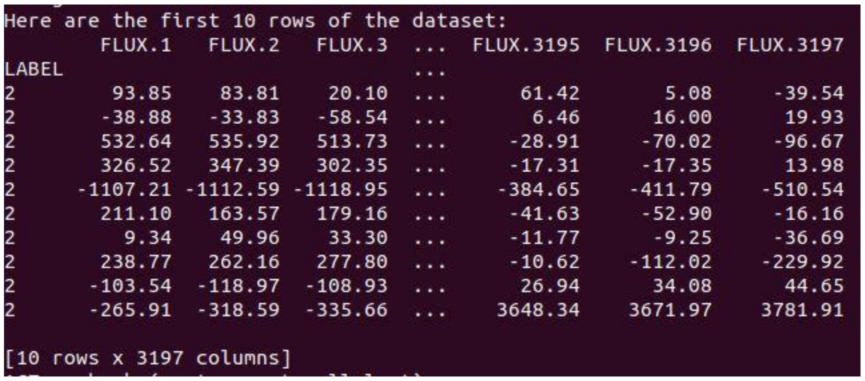
4 Preprocessing and Data Manipulation

Since all the data was numerical (refer to figure 1), there was no need to convert values to/from categorical. This was very convenient, but there were some significant adjustments needed in order to carry out the analysis. The data was split into one label attribute which was in binary format (1 for non-exoplanet star and 2 for exoplanet) and over 3000 flux values where the length in time between flux attributes was uniform among each row. In the train data set, there were 37 confirmed exoplanets and 5050 non-exoplanet stars, whereas the test data set had 5 confirmed exoplanets and 565 confirmed non-exoplanets. Before any data manipulation was

[2] DeepAI. (2019, May 17). Weight (Artificial Neural Network). Retrieved December 13, 2020, from https://deepai.org/machine-learning-glossary-and-terms/weight-artificial-neural-network

done, it was important to understand the spread of the data. Taking a quick look at the data, one will notice that most of the data was negative. However, the positive values directly corresponded to the confirmed exoplanets, so they were crucial/of high importance. To understand the spread of the data, using the matplotlib library, basic statistical curves were plotted to see any clear outliers and to understand the basic structure, refer to figure 2.

Figure 1



```
Here are the first 10 rows of the dataset:
        FLUX.1   FLUX.2   FLUX.3  ...  FLUX.3195  FLUX.3196  FLUX.3197
LABEL                             ...
2        93.85    83.81    20.10  ...      61.42       5.08     -39.54
2       -38.88   -33.83   -58.54  ...       6.46      16.00      19.93
2       532.64   535.92   513.73  ...     -28.91     -70.02     -96.67
2       326.52   347.39   302.35  ...     -17.31     -17.35      13.98
2     -1107.21 -1112.59 -1118.95  ...    -384.65    -411.79    -510.54
2       211.10   163.57   179.16  ...     -41.63     -52.90     -16.16
2         9.34    49.96    33.30  ...     -11.77      -9.25     -36.69
2       238.77   262.16   277.80  ...     -10.62    -112.02    -229.92
2      -103.54  -118.97  -108.93  ...      26.94      34.08      44.65
2      -265.91  -318.59  -335.66  ...    3648.34    3671.97    3781.91

[10 rows x 3197 columns]
```

Figure 1 shows output of what the first 10 rows of the data look like. Note that there are over 3000 attributes and each value is numeric. Also note that each of the rows shown correspond to an exoplanet as the label is of value 2.

Figure 2

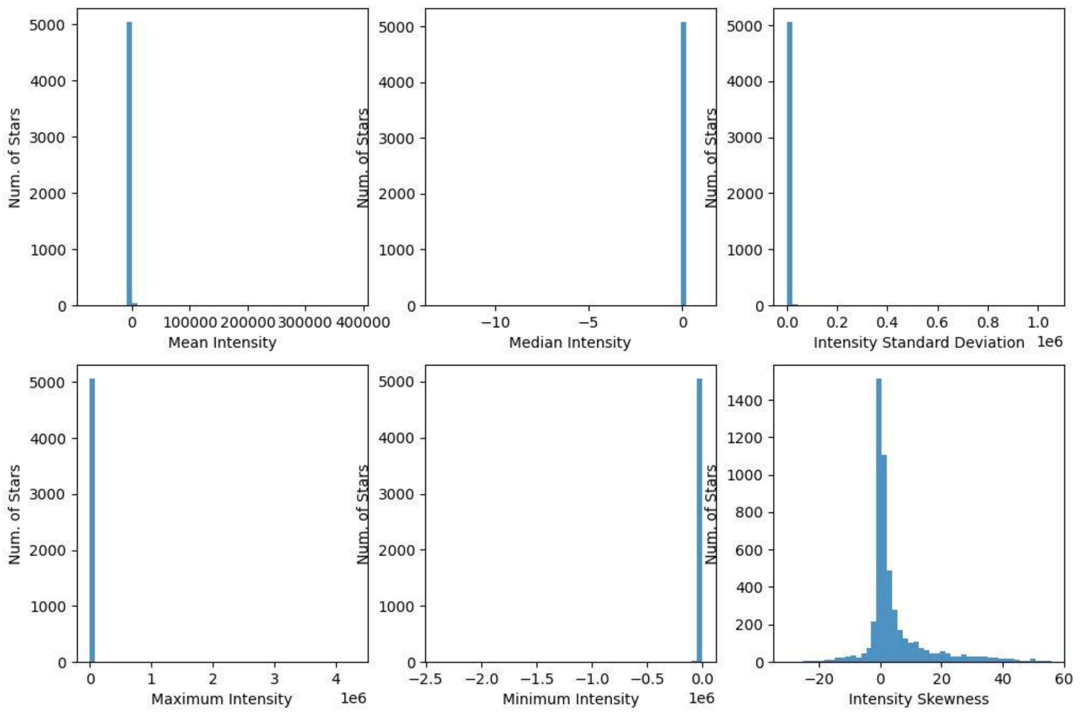


Figure 2 some of the basic statistical curves of the data, which was used to determine outliers and specific techniques of preprocessing needed. There are significant outliers and that the data has heavy emphasis on values close to 0, so balancing the data is crucial.

It was immediately clear that the data needed to be significantly balanced. There was heavy emphasis on values close to 0 and the outliers were significant. Additionally, before any preprocessing, it was helpful to understand the spread of the flux values. Using the same library (matplotlib), a graph of the spread of the 37 confirmed exoplanets was compared with 37 random non-exoplanets to see if there was any clear correlation to base the preprocessing on, refer to figure 3.

Figure 3

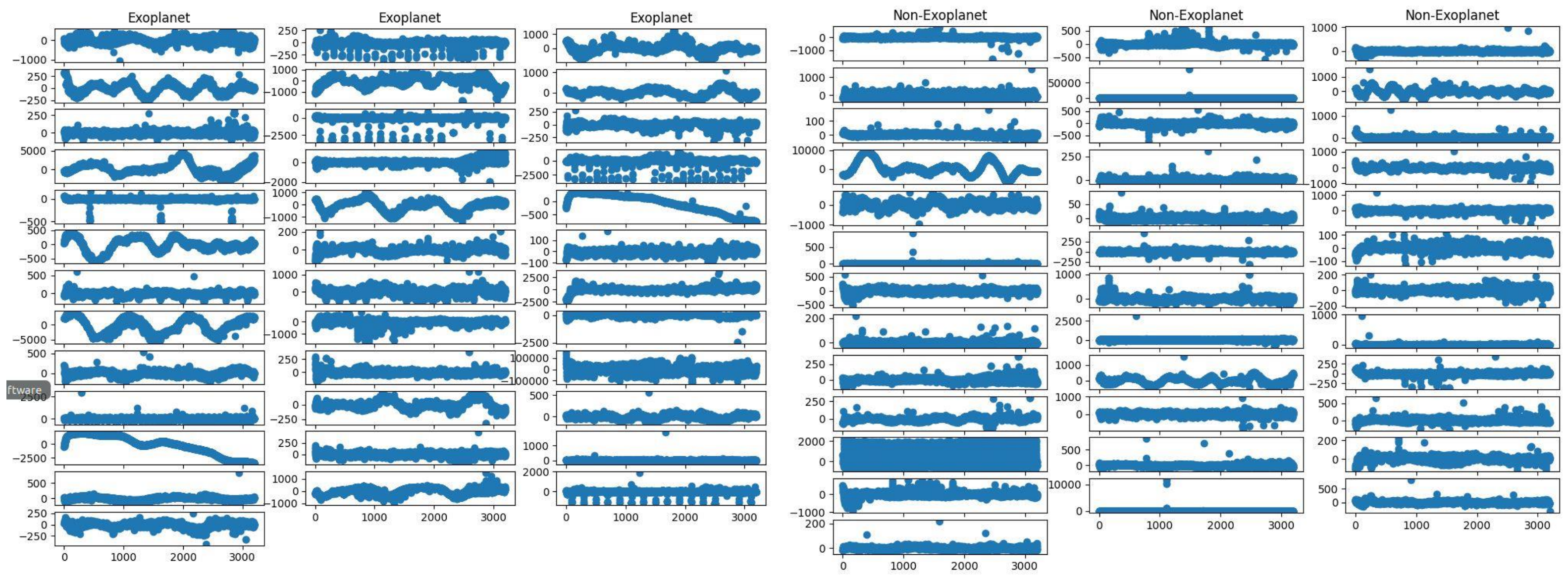
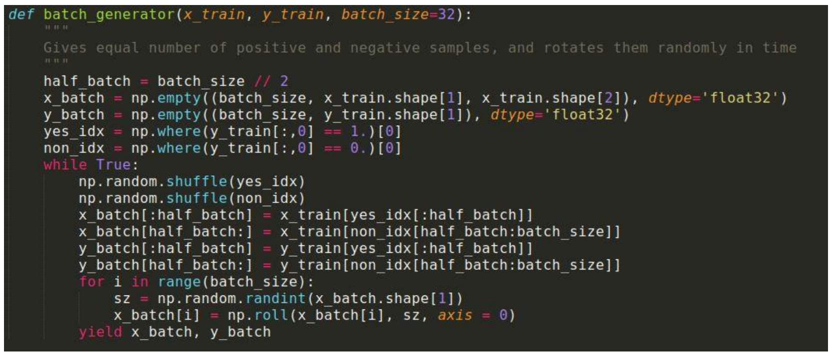


Figure 3 shows a direct comparison to 37 flux values of exoplanets (on the left half) to 37 flux values of non-exoplanets (right half). This was done with the hopes of finding some sort of correlation between the distribution of the light intensities. Unfortunately, there is nothing sticking out, so more in-depth preprocessing techniques were clearly needed. The x-axis represents the flux number attribute and the u-axis represents the corresponding flux value for that attribute.

Even looking very closely at the flux graphs, there was no indication that an exoplanet showed a specific pattern in flux values, so the preprocessing must involve significant balancing to remove the uncertainty. To do this, correcting the data to have a normal distribution would have significant benefits to the non-parametric KNN Test approach as this would significantly reduce the curse of dimensionality. Through the scipy library, the uniform_filter1d function enabled the data to appear in a normal distribution form. Now that the data was corrected to remove the significant outliers, there was still a problem dealing with the dominating number of negative flux values. So, a batch generator function was developed to balance the positive and negative values as the positive values were extremely important. This would come in handy when using the 1D CNN as each iteration would allow for learning over an equal number of positive and negative samples. To do this, the function corrects the unbalanced data by shuffling through a random portion of negative and positive values so that each iteration had a balanced agenda. Refer to the code in figure 4 to

Figure 4



```
def batch_generator(x_train, y_train, batch_size=32):
    """
    Gives equal number of positive and negative samples, and rotates them randomly in time
    """
    half_batch = batch_size // 2
    x_batch = np.empty((batch_size, x_train.shape[1], x_train.shape[2]), dtype='float32')
    y_batch = np.empty((batch_size, y_train.shape[1]), dtype='float32')
    yes_idx = np.where(y_train[:,0] == 1.)[0]
    non_idx = np.where(y_train[:,0] == 0.)[0]
    while True:
        np.random.shuffle(yes_idx)
        np.random.shuffle(non_idx)
        x_batch[:half_batch] = x_train[yes_idx[:half_batch]]
        x_batch[half_batch:] = x_train[non_idx[half_batch:batch_size]]
        y_batch[:half_batch] = y_train[yes_idx[:half_batch]]
        y_batch[half_batch:] = y_train[non_idx[half_batch:batch_size]]
        for i in range(batch_size):
            sz = np.random.randint(x_batch.shape[1])
            x_batch[i] = np.roll(x_batch[i], sz, axis = 0)
        yield x_batch, y_batch
```

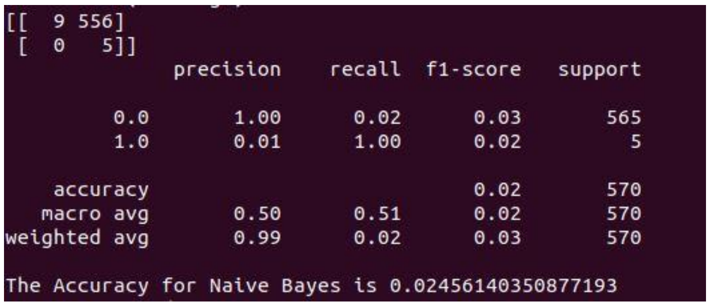
Figure 4 lists the code used to balance the data showing an equal number of positive and negative values when iterating through the datasets.

see the batch generator function[3]. After the data was balanced, it was time to begin applying the machine learning algorithms.

5 Naïve Bayes Classification Analysis

The fastest machine learning algorithm used when analyzing this dataset was by far the Naïve Bayes approach. This was because there was a significant assumption that the attributes are independent of each other. Because this assumption is inevitably alarming, as the location of an exoplanet was directly correlated on the dependence of the attributes, this method should not offer accurate results by hypothesis. This approach used the sklearn library as stated earlier. And as assumed, incredibly inaccurate results were shown, refer to figure 5 to see the output when running the Naïve Bayes Classifier.

Figure 5



```
[[  9 556]
 [  0   5]]
              precision    recall  f1-score   support

         0.0       1.00      0.02      0.03       565
         1.0       0.01      1.00      0.02         5

    accuracy                           0.02       570
   macro avg       0.50      0.51      0.02       570
weighted avg       0.99      0.02      0.03       570

The Accuracy for Naive Bayes is 0.02456140350877193
```

Figure 5 illustrates the output of the Naïve Bayes Classification method. Both the confusion matrix and the classification report are shown, with an overall accuracy of around 2.5%.

The confusion matrix describes the performance of a classification model, on a set of test data where the true values are known. In this case, it was known that 37 exoplanets were confirmed in the test dataset, so this is where the Naïve Bayes Classifier predicted its results. The values in the confusion matrix are as follows (using matrix notation where ab = row a, column b): the 00 location describes the true positive value, which is the number of points that the classifier correctly predicted as positive. The 01 location describes the false positive value, which is the number of points the classifier predicts to be positive when they belong to the negative class. The 11 location describes the true negative value, which is the number of points that the classifier correctly predicted as negative. Lastly, the 10 location describes the false negative value, which is the number of points that the classifier predicts to be negative when they are positive. With this understanding there were a significant number of false positives, in fact 556 false positives. The classification report allows for other insight into the efficiency and accuracy of the classification. The accuracy of this classifier is the fraction of the correct predictions over all points predicted to be in the class. The recall is the fraction of correct predictions over all points in the class. Lastly, the F1-score balances the precision and recall values by computing their harmonic mean for the class. Now, looking at the classification report, the calculation of the accuracy was very low with an overall score of around 2.5%. This makes the Naïve Bayes an unreasonable approach to predicting exoplanet locations from these datasets. This makes complete sense as the assumption was too significant for this type of data. Even though the output was incredibly inaccurate, it provides insight into what is needed to

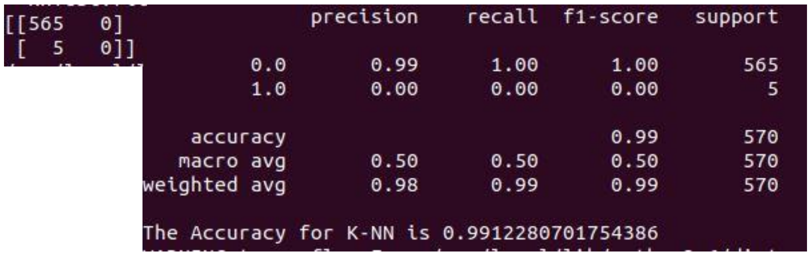
[3] Toregil. (2017, June 27). Mystery Planet (99.8% CNN). Retrieved December 13, 2020, from https://www.kaggle.com/toregil/mystery-planet-99-8-cnn

perform a better analysis. In addition, it was foundational in understanding that a non-parametric approach like the KNN test should yield better results.

6 K Nearest Neighbors (KNN) Test

Like the Naïve Bayes Classification technique, the KNN test relied on the sklearn library. After modifying the data so that it appeared to have a normal distribution, the KNN Test should offer significant accuracy as the KNN Test relies on distances, meaning that the smaller the dimensionality, the better the results, refer to figure 6 to see the results in the output.

Figure 6



```
[[565   0]
 [  5   0]]
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00       565
         1.0       0.00      0.00      0.00         5

    accuracy                           0.99       570
   macro avg       0.50      0.50      0.50       570
weighted avg       0.98      0.99      0.99       570

The Accuracy for K-NN is 0.9912280701754386
```

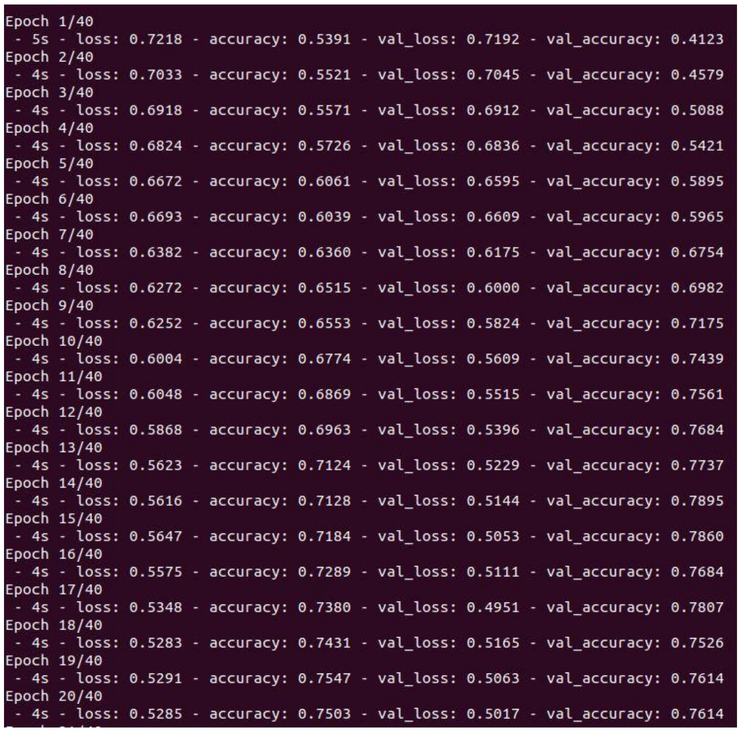
Figure 6 illustrates the output of the KNN Test method. Both the confusion matrix and the classification report are shown, with an overall accuracy of around 99%.

The output created was in the exact same form as the Naïve Bayes approach because the goal was to compare a parametric to a non-parametric approach directly. Because there was no assumption on independence of the attributes, this method should have yielded results significantly better than those of the Naïve Bayes. And with the normalization of the data, the curse of dimensionality should disappear. Refer to section 5 for a description of the confusion matrix and the classification report. As assumed, the KNN Test was exponentially more accurate than the Naïve Bayes approach, leaving an accuracy score of around 99%. This was much better than expected and posed skepticism because of the incredible score. To re-evaluate if this score was valid, the program was run several times and the accuracy score averaged out to around 98%, so this score was in fact valid. The normalization of the data was crucial in getting a score this high. Even though this score was significantly greater than the Naïve Bayes, it took nearly triple the amount of time to run, however, this added amount of time was worth it as the results were phenomenal, to say the least.

7 1D Convolutional Neural Network

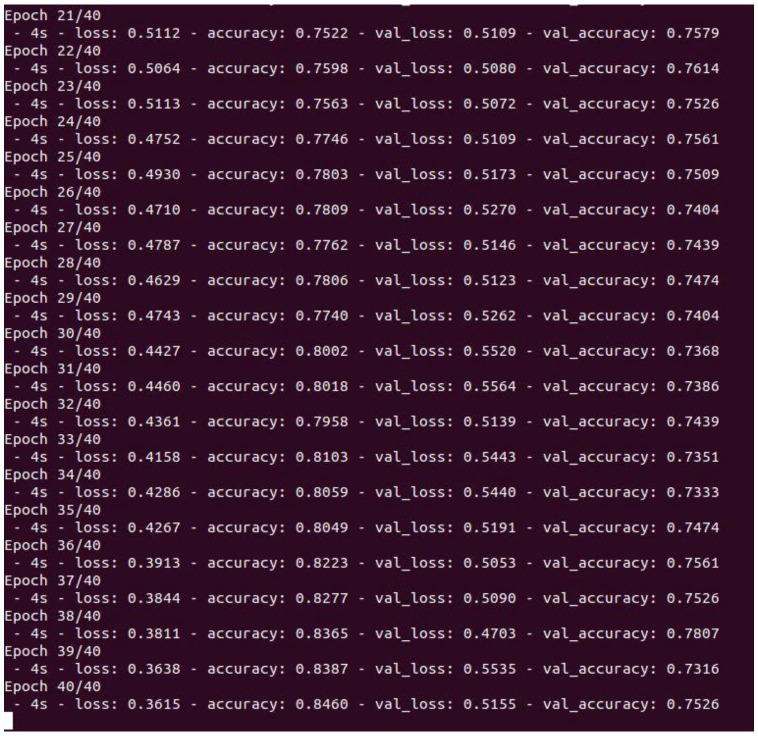
The foundation of this analysis was based on the deep learning algorithm using a 1D Convolutional Neural Network. Because the Keras library offered significant reinforcement, the code was mostly easy to write, however, due to the unbalanced nature of the data, it was very difficult to train, unlike the previous methods. As described earlier, in order to train properly, each iteration through the data must have an equal number of positive values as negative. Again, this was because the positive values were associated with the exoplanets. Using the batch generator function shown in figure 4, each run through the data would offer a random amount of positive and negative input, hopefully allowing for a convergence onto a high accuracy value. Before training the CNN at a high learning rate, it was tested with 5 epochs, or runs through the entire data set, to test that the outputs were converging correctly. Meaning, that as the iterations increased, the accuracy raised over time. After the CNN was calibrated properly using 5 epochs, it was finally run with a much higher learning rate with 40 epochs. The results of the epochs can be seen in figures 7 and 8. The output of the CNN was formatted so it would show the time each iteration took, in addition to showing the loss and the accuracy with their corresponding values. The val_loss value corresponds to the value of the cost function of the cross-validation data, whereas the loss is the value of the cost function for the training data[4]. The val_accuracy refers to a set of samples that was not shown to the network during training and hence refers to how much the model works for general cases outside the training set. On the other hand, the accuracy refers to the accuracy of the network after all epochs prior run against the test data set[5].

Figure 7



```
Epoch 1/40
 - 5s - loss: 0.7218 - accuracy: 0.5391 - val_loss: 0.7192 - val_accuracy: 0.4123
Epoch 2/40
 - 4s - loss: 0.7033 - accuracy: 0.5521 - val_loss: 0.7045 - val_accuracy: 0.4579
Epoch 3/40
 - 4s - loss: 0.6918 - accuracy: 0.5571 - val_loss: 0.6912 - val_accuracy: 0.5088
Epoch 4/40
 - 4s - loss: 0.6824 - accuracy: 0.5726 - val_loss: 0.6836 - val_accuracy: 0.5421
Epoch 5/40
 - 4s - loss: 0.6672 - accuracy: 0.6061 - val_loss: 0.6595 - val_accuracy: 0.5895
Epoch 6/40
 - 4s - loss: 0.6693 - accuracy: 0.6039 - val_loss: 0.6609 - val_accuracy: 0.5965
Epoch 7/40
 - 4s - loss: 0.6382 - accuracy: 0.6360 - val_loss: 0.6175 - val_accuracy: 0.6754
Epoch 8/40
 - 4s - loss: 0.6272 - accuracy: 0.6515 - val_loss: 0.6000 - val_accuracy: 0.6982
Epoch 9/40
 - 4s - loss: 0.6252 - accuracy: 0.6553 - val_loss: 0.5824 - val_accuracy: 0.7175
Epoch 10/40
 - 4s - loss: 0.6004 - accuracy: 0.6774 - val_loss: 0.5609 - val_accuracy: 0.7439
Epoch 11/40
 - 4s - loss: 0.6048 - accuracy: 0.6869 - val_loss: 0.5515 - val_accuracy: 0.7561
Epoch 12/40
 - 4s - loss: 0.5868 - accuracy: 0.6963 - val_loss: 0.5396 - val_accuracy: 0.7684
Epoch 13/40
 - 4s - loss: 0.5623 - accuracy: 0.7124 - val_loss: 0.5229 - val_accuracy: 0.7737
Epoch 14/40
 - 4s - loss: 0.5616 - accuracy: 0.7128 - val_loss: 0.5144 - val_accuracy: 0.7895
Epoch 15/40
 - 4s - loss: 0.5647 - accuracy: 0.7184 - val_loss: 0.5053 - val_accuracy: 0.7860
Epoch 16/40
 - 4s - loss: 0.5575 - accuracy: 0.7289 - val_loss: 0.5111 - val_accuracy: 0.7684
Epoch 17/40
 - 4s - loss: 0.5348 - accuracy: 0.7380 - val_loss: 0.4951 - val_accuracy: 0.7807
Epoch 18/40
 - 4s - loss: 0.5283 - accuracy: 0.7431 - val_loss: 0.5165 - val_accuracy: 0.7526
Epoch 19/40
 - 4s - loss: 0.5291 - accuracy: 0.7547 - val_loss: 0.5063 - val_accuracy: 0.7614
Epoch 20/40
 - 4s - loss: 0.5285 - accuracy: 0.7503 - val_loss: 0.5017 - val_accuracy: 0.7614
```

Figure 8



```
Epoch 21/40
 - 4s - loss: 0.5112 - accuracy: 0.7522 - val_loss: 0.5109 - val_accuracy: 0.7579
Epoch 22/40
 - 4s - loss: 0.5064 - accuracy: 0.7598 - val_loss: 0.5080 - val_accuracy: 0.7614
Epoch 23/40
 - 4s - loss: 0.5113 - accuracy: 0.7563 - val_loss: 0.5072 - val_accuracy: 0.7526
Epoch 24/40
 - 4s - loss: 0.4752 - accuracy: 0.7746 - val_loss: 0.5109 - val_accuracy: 0.7561
Epoch 25/40
 - 4s - loss: 0.4930 - accuracy: 0.7803 - val_loss: 0.5173 - val_accuracy: 0.7509
Epoch 26/40
 - 4s - loss: 0.4710 - accuracy: 0.7809 - val_loss: 0.5270 - val_accuracy: 0.7404
Epoch 27/40
 - 4s - loss: 0.4787 - accuracy: 0.7762 - val_loss: 0.5146 - val_accuracy: 0.7439
Epoch 28/40
 - 4s - loss: 0.4629 - accuracy: 0.7806 - val_loss: 0.5123 - val_accuracy: 0.7474
Epoch 29/40
 - 4s - loss: 0.4743 - accuracy: 0.7740 - val_loss: 0.5262 - val_accuracy: 0.7404
Epoch 30/40
 - 4s - loss: 0.4427 - accuracy: 0.8002 - val_loss: 0.5520 - val_accuracy: 0.7368
Epoch 31/40
 - 4s - loss: 0.4460 - accuracy: 0.8018 - val_loss: 0.5564 - val_accuracy: 0.7386
Epoch 32/40
 - 4s - loss: 0.4361 - accuracy: 0.7958 - val_loss: 0.5139 - val_accuracy: 0.7439
Epoch 33/40
 - 4s - loss: 0.4158 - accuracy: 0.8103 - val_loss: 0.5443 - val_accuracy: 0.7351
Epoch 34/40
 - 4s - loss: 0.4286 - accuracy: 0.8059 - val_loss: 0.5440 - val_accuracy: 0.7333
Epoch 35/40
 - 4s - loss: 0.4267 - accuracy: 0.8049 - val_loss: 0.5191 - val_accuracy: 0.7474
Epoch 36/40
 - 4s - loss: 0.3913 - accuracy: 0.8223 - val_loss: 0.5053 - val_accuracy: 0.7561
Epoch 37/40
 - 4s - loss: 0.3844 - accuracy: 0.8277 - val_loss: 0.5090 - val_accuracy: 0.7526
Epoch 38/40
 - 4s - loss: 0.3811 - accuracy: 0.8365 - val_loss: 0.4703 - val_accuracy: 0.7807
Epoch 39/40
 - 4s - loss: 0.3638 - accuracy: 0.8387 - val_loss: 0.5535 - val_accuracy: 0.7316
Epoch 40/40
 - 4s - loss: 0.3615 - accuracy: 0.8460 - val_loss: 0.5155 - val_accuracy: 0.7526
```

Figures 7 and 8 show the output of the 1D CNN when run with 40 epochs, or complete cycles through the dataset. The output was formatted so that it shows the iteration/epoch number, the time it took to complete the iteration, in addition to the loss and accuracy values. Brief scanning of the output shows that the loss decreased, and the accuracy increased over time, which was the goal.

Clearly, as the further the training went, the better the accuracy score and the lower the loss score. This was exactly the goal. Now that the iterations appeared to converge in the proper way, some analysis on the output was fundamental into understanding what the deep learning algorithm did. To start, graphs of the convergence were necessary to see if there was any relationship over time, and to what the loss and accuracy converged to, respectively. As shown in figures 9 and 10, it appears that the loss values had a negative slope and the accuracy had a positive slope as the iterations progressed. The graph was completed using the same matplotlib library as

[4] Keras difference beetween val_loss and loss during training. (1967, January 01). Retrieved December 13, 2020, from https://datascience.stackexchange.com/questions/25267/keras-difference-beetween-val-loss-and-loss-during-training

[5] J. M. (1967, August 01). Keras: How come 'accuracy' is higher than 'val_acc'? Retrieved December 13, 2020, from https://stackoverflow.com/questions/51335133/keras-how-come-accuracy-is-higher-than-val-acc

used before. The result was exactly as expected. After the convergence was verified, the analysis of the effectiveness of the algorithm commenced. First, it was important to understand the values of prediction that the

Figure 9

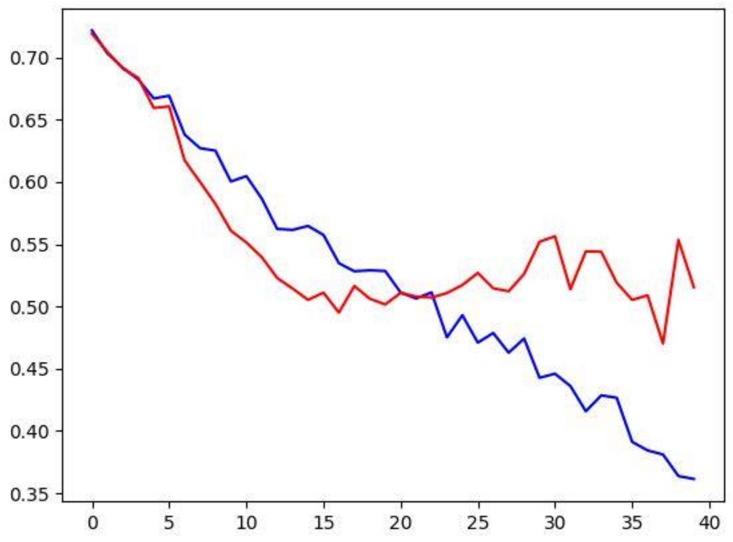
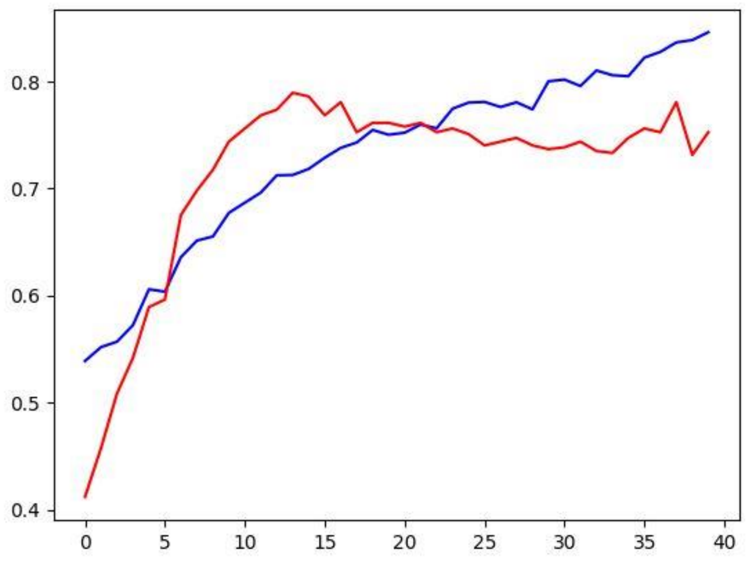


Figure 10



Figures 9 and 10 show the convergence of the 1D CNN over time. Figure 9 is the convergence of the loss whereas figure 10 is the convergence of accuracy. The x-axis in both figures represents the epoch number where the y-axis represents the percentage score. The blue curve represents the loss and accuracy, respectively, and the red curve represents the val_loss and val_accuracy, respectively.

exoplanets received and compared that to the outputs that the non-exoplanets received. To do this, all that was needed was to plot the true cases, being the values when the label attribute was 2. So, after the completion of all 40 epochs, the prediction values corresponding to each exoplanet were drawn (matplotlib) as shown in figure 11.

Figure 11

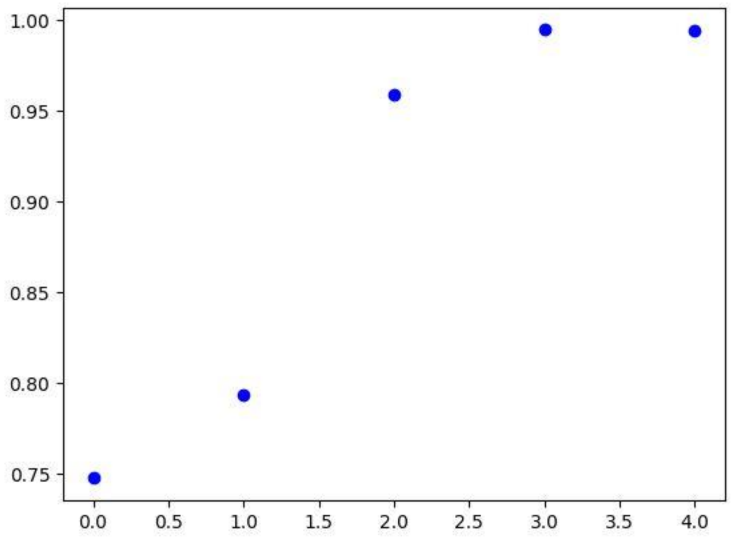


Figure 11 shows the prediction values for all 5 of the confirmed exoplanets from the test set. It is very important to note that all the values received a score of 75% or better, with the majority having a score of over 95%. This is relieving as the algorithm correctly identified the exoplanets!

Looking at figure 11, one can note that all the prediction scores for the confirmed exoplanets from the test set had a value over 75%. In fact, a majority had a value over 95%. This means that the algorithm did a great job at identifying the true exoplanets and scored them with a high chance of being an exoplanet. This was great to see as it appears that the algorithm worked. Now, it was very important to see what prediction values the non-exoplanets got, as this could shed some light into the error in the algorithm. Using the same graphing technique as in figure 11, figure 12 shows the prediction percentage outputs for all the non-exoplanets. By assumption, these values should be as close to 0 as possible, as they are confirmed to be non-exoplanets. However, looking at figure 12 shows that this was not the case.

Figure 12

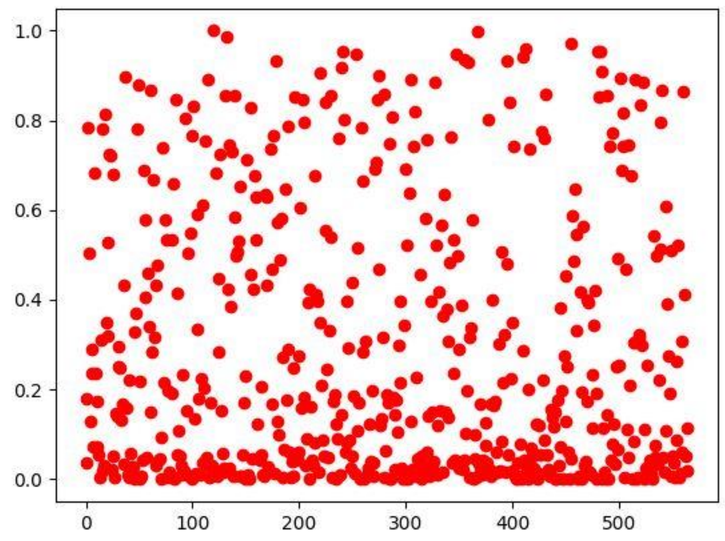


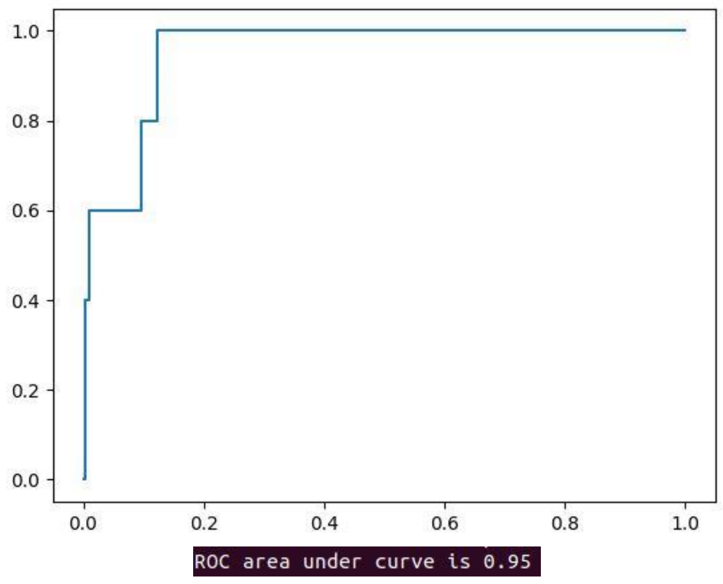
Figure 11 shows the prediction values for all 565 of the confirmed non-exoplanets from the test set. It is very important to note that although most of the values are close to 0, there is a significant amount that fall above 50%, which is concerning as it implies the algorithm mis-identified non-exoplanets as exoplanets.

Although most of the values appear to be close to 0, as expected, there are still a significant amount that received a score of over 50%. This is alarming as the algorithm clearly mis-interpreted and falsely identified a non-exoplanet as an exoplanet. This means that all the values greater than 50% were in fact false positives. So, although the algorithm did an excellent job at identifying the correct exoplanets, it did not do well in identifying non-exoplanets. To see how badly this algorithm misclassified non-exoplanets, ROC curve and cutoff analysis was performed. Firstly, ROC curve analysis plots the performance of the classifier over all possible values of a threshold parameter. For each threshold value, a ROC curve plots the false positive rate vs. the true positive rate. The ideal classifier should score all positive points (correctly identified points) higher than any negative (misidentified points), meaning that the ROC curve has an area of 1. The higher the area under the ROC curve, the better the classifier. This area value is essentially the probability that the classifier will rank a random positive test case higher than a random negative test instance[6]. Plotting this ROC curve was basic as the sklearn

[6] Zaki, M. J., & Meira, W. (2020). *Data Mining and Machine Learning: Fundamental Concepts and Algorithms*. Cambridge: Cambridge University Press.

library has a roc_curve splitter function. The output and result of the ROC curve can be seen in figure 13, with the corresponding area.

Figure 13



```
ROC area under curve is 0.95
```

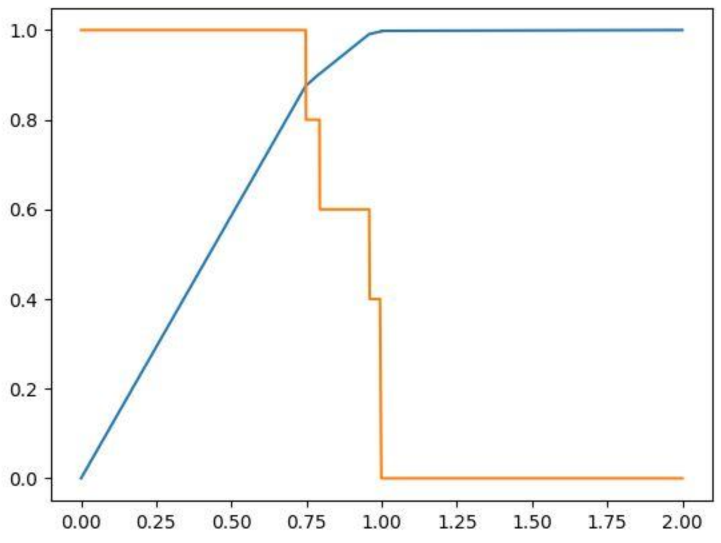
Figure 13 shows ROC curve with the output including the area under the curve. It is important to note that the closer the area is to 1, the better the classifier. Since the area was .95, that is notably good considering the misidentification of exoplanets shown earlier.

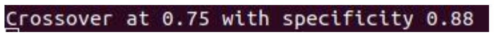
Even though there were notable false positives, the ROC curve still gave an area of .95, which is very close to 1 making it a high performing classifier. In addition to the ROC curve, cutoff analysis was also used to verify the performance of the deep learning algorithm. The ROC curve can also be described as 1 – specificity. The specificity is the true negative rate, or the recall for the negative class. The optimal cutoff of the ROC curve is defined to be the optimal cut-point value as the point minimizing the summation of absolute values of the differences between the area under the AUC (area under the ROC curve) and sensitivity with the AUC and specificity, provided that the difference between sensitivity and specificity is minimum[7]. Applying this to the ROC curve that was already generated was also straight forward as the crossover index value can be solved just using numpy's min function. The output and results of the optimal cutoff analysis can be seen in figure 14. Note that the specificity is the true negative rate and therefore, a higher value implies a better identification of values in the negative (non-expolanet) class. This is equivalent to saying that the higher the specificity, the better the error rate. The specificity can be seen to be equal to .88 which is good considering that the remaining 12% indicates that there were around 12% false positives. From figure 12, this was already determined. Lastly, the false positive values need to be determined. This was straight forward because all we needed to do was identify the points in figure 12 that had a score greater than 50% because then it would have been misidentified, as the points in figure 12 were all confirmed non-exoplanets. Looping through the entire set, proved to yield quite a bit of false positive values. In fact, after running the program several times it appeared that there was consistently between 20 and 50 non-exoplanets that would be misidentified. Out of the 560, 50 is quite a large portion to

[7] Unal, I. (2017, May 31). Defining an Optimal Cut-Point Value in ROC Analysis: An Alternative Approach. Retrieved December 13, 2020, from https://www.hindawi.com/journals/cmmm/2017/3762651/

have misidentified. To understand where the misidentification occurred, a graph of each false positive star with the

Figure 14





```
Crossover at 0.75 with specificity 0.88
```

Figure 14 shows the result of the optimal cutoff score from the ROC curve. It also shows the output of the code indicating the crossover at .75 with specificity .88.

flux values in addition to where the misclassification occurred was illustrated. Because there was averaging between 20 to 50 false positive values, for clear formatting, 5 of the misclassification graphs can be seen in the appendix in figures 15-19. Although the deep learning algorithm falsely identified a somewhat significant fraction of the stars, it proved to be incredibly accurate over identifying exoplanets. The strenuousness of the algorithm came from the unbalanced nature of the data. Better balancing will inevitably solve this false positive issue.

8 Results

Direct comparison between the algorithms shows that the KNN Test was clearly the best at predicting exoplanets. The Naïve Bayes approach having an accuracy score of around 2.5% offers little support and evidence in discovering new exoplanet locations. The 1D CNN proved to be much better with an accuracy score that converged to over 85%, however, there were several misclassified stars that threatens the credibility of the algorithm. Lastly, the KNN Test proved to be most suitable for this dataset as the non-parametric approach combined with the normalization of the data yielded in a near perfect classification with an average accuracy score (over multiple runs) of 98%.

9 Limitations and Error

It is inappropriate to assume perfect accuracy. A widespread distribution of accuracy values was shown through the 3 machine learning techniques. It can be concluded that the naïve assumption of independence between attributes is not feasible. The key factor in the KNN Test to have such a high score was due to the normalization of the data basically removing the curse of dimensionality. The error in this test still came from this attempt to balance the data via normalization. Although the error for this method was insignificant compared to the other techniques, it was still relevant as it was not a perfect classification. Additionally, if the data could have been balanced better, the neural network could have achieved a higher accuracy score without the false positive values it generated. Nearly all the error in this project came from the attempt to balance the data. Because the data was so skewed, it was difficult to come up with a process or technique that would allow for a random distribution of input stars while maintaining the positive importance. The batch generator function that can be seen in figure 4 proved to be functional, however, this is where the misclassification occurred from the 1D CNN. As this is an ongoing Nasa mission, the identification of exoplanets through this data set can only be improved with an increased ability to preprocess and manipulate this data into a balanced form.

10 What's Next

As stated earlier, this is one of Nasa's ongoing missions to hunt for exoplanets based on these light flux changes. Although no new planet was discovered in this analysis, the methods and techniques prove to give insight into what works and what doesn't. It seems like developers need to focus their time on the preprocessing, and once a balanced dataset can be formed the non-parametric approach should yield the best results.

A Appendix

Figure 15

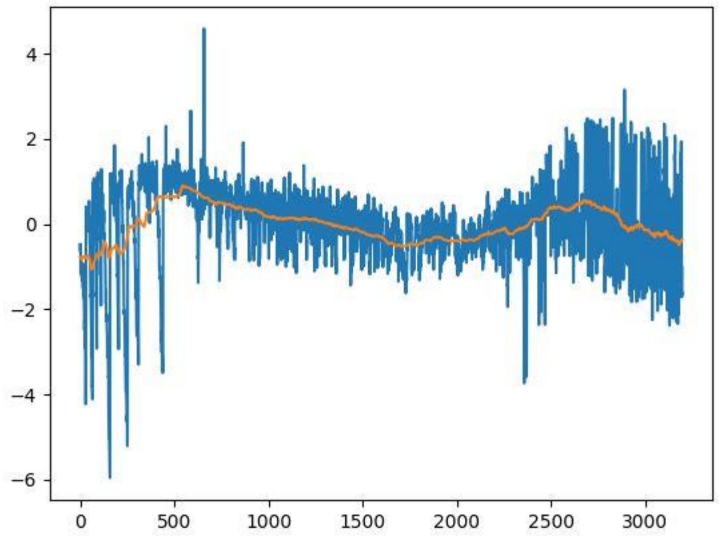


Figure 15 shows the misclassification of the 1st non-exoplanet star which can be identified as the 7th row of the dataset.

Figure 16

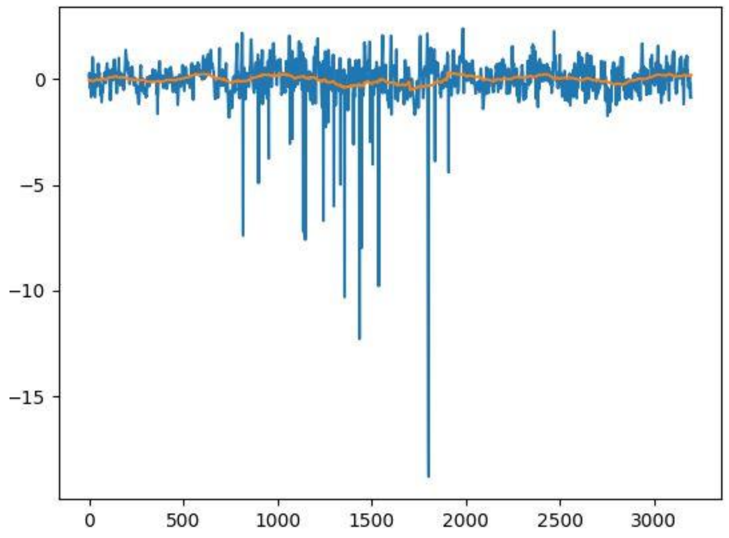


Figure 16 shows the misclassification of the 2nd non-exoplanet star which can be identified as the 21st row of the dataset.

Figure 17

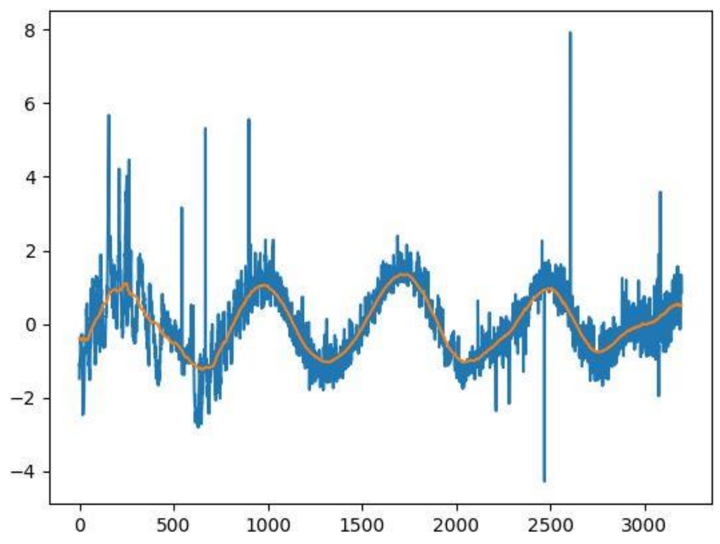


Figure 17 shows the misclassification of the 3rd non-exoplanet star which can be identified as the 23rd row of the dataset.

Figure 18

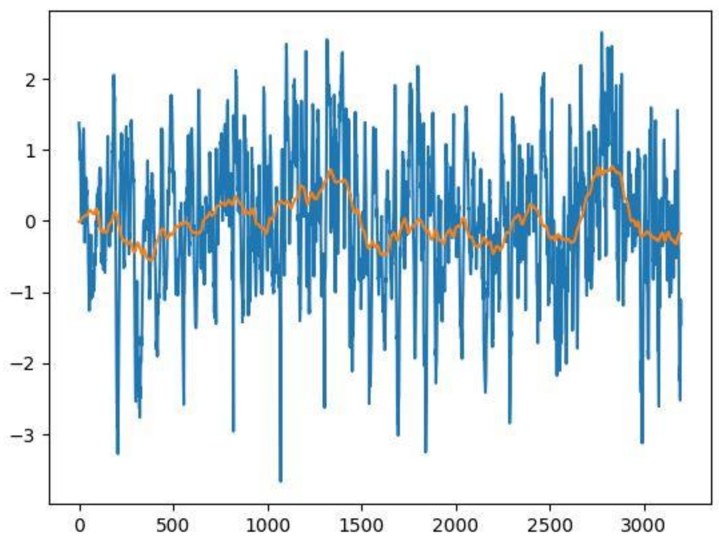


Figure 18 shows the misclassification of the 4th non-exoplanet star which can be identified as the 54th row of the dataset.

Figure 19

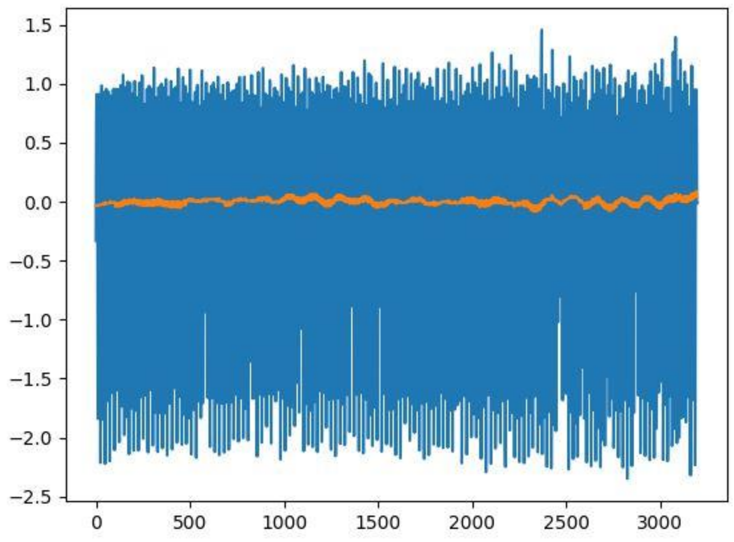


Figure 19 shows the misclassification of the 5th non-exoplanet star which can be identified as the 124th row of the dataset.

B References

Zaki, M. J., & Meira, W. (2020). *Data Mining and Machine Learning: Fundamental Concepts and Algorithms*. Cambridge: Cambridge University Press.

WΔ. (2017, April 12). Exoplanet Hunting in Deep Space. Retrieved December 12, 2020, from https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data

Toregil. (2017, June 27). Mystery Planet (99.8% CNN). Retrieved December 13, 2020, from https://www.kaggle.com/toregil/mystery-planet-99-8-cnn

DeepAI. (2019, May 17). Weight (Artificial Neural Network). Retrieved December 13, 2020, from https://deepai.org/machine-learning-glossary-and-terms/weight-artificial-neural-network

Solutions, E., & Name, *. (2016, November 11). Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures. Retrieved October 31, 2020, from https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/

(n.d.). Retrieved December 13, 2020, from https://raw.githubusercontent.com/vivek2319/Predicting-US-Census-Income/master/predict.py

Solutions, E., & Name, *. (2016, November 11). Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures. Retrieved October 31, 2020, from https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/

Sklearn.metrics.precision_recall_fscore_support¶. (n.d.). Retrieved October 31, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html

Precision and recall. (2020, October 14). Retrieved October 31, 2020, from https://en.wikipedia.org/wiki/Precision_and_recall

Classification: Accuracy | Machine Learning Crash Course. (n.d.). Retrieved October 31, 2020, from https://developers.google.com/machine-learning/crash-course/classification/accuracy

Unal, I. (2017, May 31). Defining an Optimal Cut-Point Value in ROC Analysis: An Alternative Approach. Retrieved December 13, 2020, from https://www.hindawi.com/journals/cmmm/2017/3762651/

J. M. (1967, August 01). Keras: How come 'accuracy' is higher than 'val_acc'? Retrieved December 13, 2020, from https://stackoverflow.com/questions/51335133/keras-how-come-accuracy-is-higher-than-val-acc

Keras difference beetween val_loss and loss during training. (1967, January 01). Retrieved December 13, 2020, from https://datascience.stackexchange.com/questions/25267/keras-difference-beetween-val-loss-and-loss-during-training