# Exoplanet Hunting

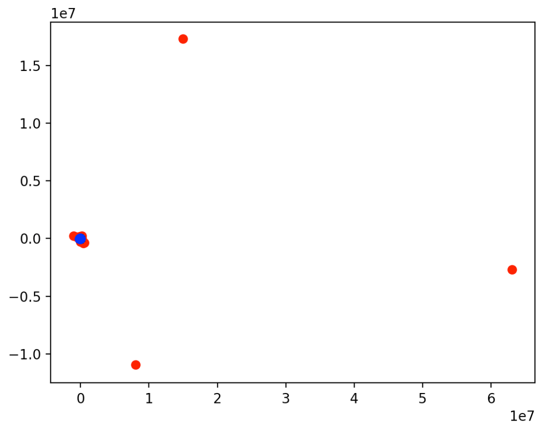
By: Rowan Lavelle and Nick Frasco

We set out to compare four different methods of data classification to decide which method would have the best classification rate of exoplanets. One of the major challenges we ran into was how unbalanced the original data was. When we found the data, there were 37 planets and 5050 non-planets. This meant that if any method were to classify all the objects as non-planets, it would achieve a success rate of 100%. To overcome this, we used an array of different data preprocessing techniques. The major one we utilized was SMOTE (Synthetic minority oversampling technique). One variation of our problem could be rather than determining if an object is an exoplanet or not, our classifiers would determine if a certain planet is habitable or not. Another variation might be classifying a certain exoplanet according to its orbit and mass. Our original idea was to have some sort of classification method be able to classify a planet (or exoplanet) as habitable or not based off of it's attributes. We soon realized that we are not astrophysicist's and the data was way out of our knowledge range. Because of this, we pivoted to just classifying objects as exoplanets or not.

An intelligent agent would use this data in the first step to finding a terrestrial planet in the milky way. The first step is finding an exoplanet, the second step is classifying that exoplanet, and the third step is determining whether or not it is terrestrial based off of mass and proximity to a hot star. After running the data through our program, an intelligent agent would then make the decision as to whether or not a certain object is able to be classified. An exoplanet can be classified as a gas giant, ice giant, or a hot super-Earth. An agent would also be able to get a light fluctuation graph from the data

**matplot –** we used matplotlib to graph and visualize the data to better understand what we were working with.

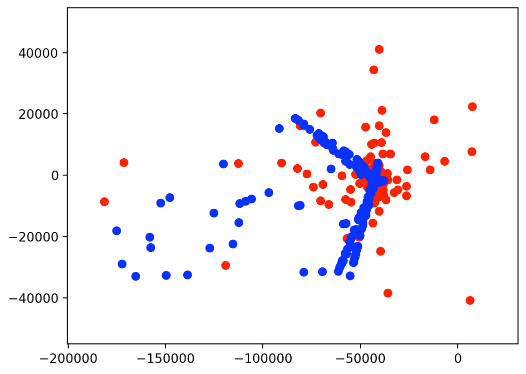
After using PCA to reduce the attributes from 3,197 to two, we used matplot to graph all of the training points. This gives us a good insight to see where the positive and negative points lie.



**NumPy –** We used numPy to store the data into inputs and outputs. The input data is the attribute and the output data is the label. The data took the form of a list of planets where for a planet the label is the first entry and the remaining attributes are light fluctuations. A label of 1 means the object is not an exoplanet and a label of 2 means the object is an exoplanet.

**Pandas** – We use Pandas to easily read in our data from a csv file in which it puts the data into a data frame and is easily accessible and organized. From the data frame we can directly pull the numPy array using "dataframe.values".

**SMOTE** – We used SMOTE to rebalance the unbalanced data. It came from "imblearn.over_sampling," using the attribute "random state = 42," this has the smote function return a perfect dataset where there are an equal number of positive and negative data points. SMOTE works by a point in the underrepresented data and draws a line to another piece of underrepresented data and populates that line with fake underrepresented data.



**Scikit-learn –**

**Standard scalar –** data preprocessing method which takes in an array of data and returns an array of the same data but fit to a standard Gaussian distribution.

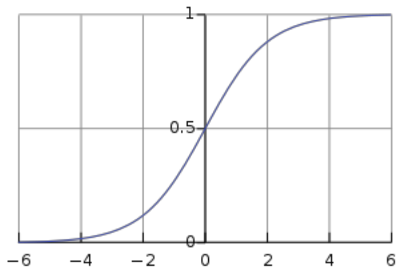
**Normalizer –** Normalizer takes in a list of data and returns the same list of data in unit form. Unit form means it has a magnitude of one.

**PCA** – We used Principal Component Analysis to cut down the attributes in the data in order to graph.

**Job-lib** – job lib assisted us in saving and reusing train models by dumping the components of the models into pickle files.

**Classifiers** –

**MLPClassifier –** Also known as Multi-Layer Perceptron Classifier, our neural network had four layers; an input layer, two hidden layers, and an output layer. The input layer had 3,197 nodes, the two hidden layers had 800 nodes each, and the output node had one. We used the built in "adam," solver which is a gradient decent based optimizer. We used a sigmoid function as our activation function which is defined by $f(x) = \frac{1}{1+e-x}$. We used a learning rate of .001; the learning rate of the network controls the step size to update the weights of the network. We used a batch size of 32; this means 32 training examples are fed through the perceptron at the same time.



**KNeighborsClassifier** – we used this as the model for K-Nearest-Neighbor with a value of k=11. This classifier works best using 570 training points.

**DecisionTreeClassifier –** we used this as the model for the Decision Tree and it was the worst performing model .

**Scipy –**

**Gaussian Filter –** We used the Gaussian filter to reduce the noise from our data set.

**Fourier Transformation -** Used this to decompose the light wave as a signal into more rich and discriminative features.

**Self K-Nearest Neighbor** – The self K nearest neighbor had a runtime of $O(n * m^3 * a)$ Where n is amount of testing points, m is the amount of training points, and a is the amount of attributes. A high level description of the algorithm is as follows;

1. Load in data from preprocessing class
2. For each testing point X
    a. Get a list of distances from X to each point in training data
    b. Use bubble sort to sort list of distances from low to high
    c. Return most occurring label from 0 to K from the sorted list of distances.

**Empirical Analysis**

| **Classifier** | **Accuracy %** | **Confusion Matrix** | |
|---|---|---|---|
| Neural Network | 74 | 333 | 232 |
| | | 0 | 5 |
| K Nearest Neighbor | 93 | 498 | 67 |
| | | 1 | 4 |
| Decision Tree | 98 | 550 | 15 |
| | | 4 | 1 |
| Self K Nearest Neighbor | 89 | 463 | 102 |
| | | 0 | 5 |

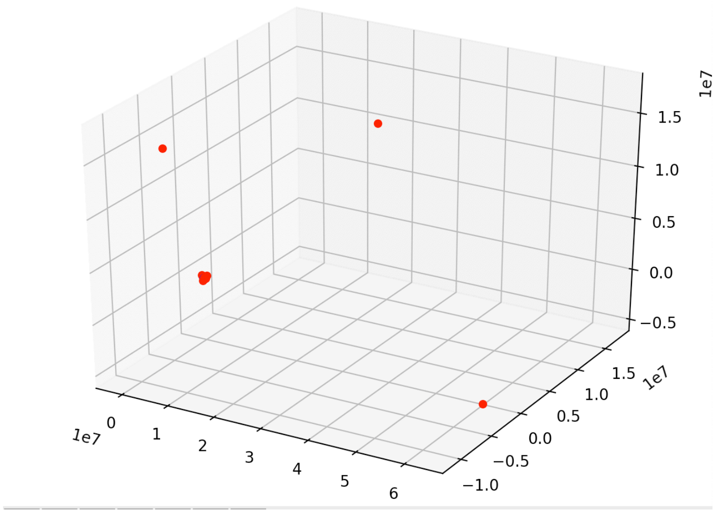
Accuracy = $2 * \{\frac{P*R}{P+R}\}$
Where P is equal to precision and R is equal to recall

Confusion matrix =

| | |
|---|---|
| True Negative | False Positive |
| False Negative | True Positive |

**Results**:

The KNN model performed best with the self KNN coming in second and the Neural Network coming in third. After reducing the attributes down to two, we found the planets were all in a cluster, which made it a hard for the neural network to learn. On the contrary this cluster explains why the nearest neighbors models performed well. We could have expanded our work into how different ways of data preprocessing affects the results of the classification by separately testing each classifier with each of the preprocessing methods and comparing the outputs. Some improvements we could make to the AI is playing around with the neural network till we get a better classification rate. We could even put it in tensor flow and see how that compares. We learned that preprocessing your data prior to using it is just as important, if not more important, than how you classify it. Without a representative sample of data, your results will be skewed and your classification rate will be false.


1e7
1.5
1.0
0.5
0.0
−0.5
1.5
1.0
0.5
0.0
−0.5
−1.0
1e7
1e7
0
1
2
3
4
5
6