Freddy Castro

DATA 71200

Using Machine Learning Algorithms to Predict Hazardous Asteroids

During the first week of class, we were grouped with our peers to discuss what we thought machine learning is and what we wish to learn as well as to discuss possible datasets we would consider for our upcoming project. I initially had no idea what machine learning was nor did I know what dataset would work best for the task at hand. All I knew about the class was that it involved math, coding, and plenty of reading. While discussing what topics to explore for our project, I did remember that I worked with space data in one of my visualization classes and I spent a good amount of time dissecting the data rich file. I had a topic down but now what was left was what portion of space do I want to explore. My curiosity was piqued when my classmate mentioned asteroids and then everything fell into place. From what I know about astronomy, most asteroids that enter our atmosphere burn up as they fall towards the Earth, some break apart, and others are durable enough to make a crater when they strike the ground. There have been plenty of headlines that I read about a possible meteor entering our atmosphere that can cause some serious damage, and that is where the idea of predicting whether an asteroid should raise some concern using basic information about that asteroid such as its diameter, orbit class, and its trajectory.

The dataset itself was taken from Kaggle and contains a plethora of information. The variable I am trying to predict from this dataset is the “PHA” variable, potentially hazardous asteroid, which is labeled as a Boolean expression, yes or no. The rest of the file contained numbers, strings, and some missing cells. Most of the data wrangling was spent on addressing missing values, adding dummy variables, deciding to whether drop a feature entirely, and changing my target variable from a Y or N label into one that python can read and predict. I focused on checking what data types my file had and then compiled a list of how many missing values each unique feature had. For missing numerical values, I decided to fill these in with their median. This approach was taken since the median is able to handle outliers in the data much better than the average and therefore not skew the end result. Missing categorical values were dropped completely along with the corresponding row since these null values had no way to be replaced or filled in. Other categorical features that described the ID of the asteroid, such as the "name" and "object ID" were dropped as well since these values had no relation to "PHA".

To address the features that were not numerical, I first separated the categorical and numerical values and stored them in their own data frames. I then found how many unique values the "orbit_id" had to address whether to drop it completely or add dummy variables to each class identifier. The "orbit_id" feature contained 526 unique values and therefore would result in 1052 additional dummy features. This was adding more data to an already data rich file and I decided to drop the "orbit_id" completely. I then labeled encoded the target variable "PHA" from N and Y to 0 and 1 so that the algorithm can read the file. After double checking that there were no missing numerical values and categorical values, and that the dummy variables were encoded, and that the target variable has been switched to 0 and 1, I merge the data back together and proceeded to do the necessary visualizations.

The first objective was to check if "PHA", the target variable, had high or low correlation to the features. The first few high correlations were the dummy variables for "NEO", near earth orbit. It seems that asteroids that are closer to earth pose a higher threat than those that are not.

The other high correlations were the asteroids eccentricity, mean motion, and orbit classification. Other visualizations such as histograms were plotted, as well as scatter plots and their corresponding matrix. Some of the data points seemed to be normally distributed or otherwise skewed to the right, but since the target variable was a yes or no answer, a classification problem, the ideal machine learning algorithm would be decision trees or logistic regression, both of which do not require to standardize our data. Since I also added dummy variables to "orbit_class", an additional 14 columns were added to my data and the graphs pictured consisted mainly of 0 or 1 values, which did not offer much information. Transforming my numerical values also did not do much to standardize the data since my file was already big enough to capture the necessary boundaries of an asteroid that would be labeled as hazardous or not.

Most of the challenge resided in the null values and finding an appropriate encoder to change my categorical values to numerical values. The next step was to split the data and run the stratified shuffle split on my immense file. This was done because our target variable, "PHA", consisted less than 1% of positive potentially hazardous asteroids. If I were to not split the data, the machine learning algorithm would have been weighted heavily on the "No" end result. The data was split ten times and then shuffled to ensure we would get an accurate reading by both logistic regression and the decision tree classifier. Since I only have two possible outcomes in my target variable, logistic regression was chosen over K-nearest neighbor. I ran the logistic regression on the split data and then adjusted C, the strength of regularization to see if there was any improvement in the test score. I initially thought that if I increased the C parameter, the test score would be lowered since we are trying to fit the training set as best as possible and we would have a stricter boundary to capture all the appropriate values that considers an asteroid hazardous or not. The score also remained high when I lowered the C values to 0.01. The scores to the ten splits for each parameter change was around 99.778% which is relatively high. I was expecting some fluctuation when running the algorithm on the test set, but it was surprisingly very accurate.

The next supervised learning algorithm I chose to run was a decision tree. I took the same approach as when I did logistic regression and used the same stratified shuffle split data and ran it through the algorithm using the default parameters. The test scores for the ten folds were all around 99.99% accurate, which almost blew me away and almost sounded too good to be true. When running the classification report on the data, we can see that precision, recall, and the F1 score were almost a perfect score. Using a decision tree was handling the data much better than logistic regression since the classification report returned nearly perfect scores when using the default parameters. Changing the max depth of the trees did not affect the score by much but using a depth of three returned values that were the closest to a perfect score, if rounded to the ten thousandths place. For a decision tree, the bulk of how it decides where to place the target variable is through a list of "true, false" statements until it arrives to the desired output. It is almost like a giant pipeline that the data must take and meet certain requirements before going on to the next series of questions. The max tree depth is the parameter that allows the algorithm to develop only up to that number of consecutive questions before the final decision is made. In my case, the max tree depth amount that produced the most accurate results were three; this was a pretty shallow tree.

Decision trees have another property that compiles a list of the most important features, and when the tree depth was 3, it placed the most importance on "H" the absolute magnitude parameter for an asteroid. Importance for this feature was 79.99%. I believe the decision tree method proved to be the most useful because it worked with all the data to create a true or false parameter. The algorithm would carefully look at each point and compare it with the rest of the data and have it travel down the leaves until it fell on the right leaf. With logistic regression, a lower C value puts more emphasis on coefficient vectors that are closer to zero, effectively erasing them from the equation.

The next project consisted of running the data using unsupervised learning. For this project we standardized our data and transformed it through PCA, principal component analysis. The PCA rotates the dataset such that the features are statistically uncorrelated. Then some features are dropped and only those that are important in explaining the data are kept. For the final project we were tasked with running our PCA transformed data on our best performing supervised learning algorithm and see if we can improve the overall score. Since my best performing algorithm scored 99.99%, I ran it on logistic regression. With the standardized and PCA transformed training set, running logistic regression to the ten folded data returned just about the same scores as using the default parameters from project 2 which was 99.77% when using PCA and 99.78% when using the default parameters. Even when going from 49 features to 27 features to capture 95% of the variance, logistic regression was still capable of producing a high score.

Prepping the data for clustering using PCA ran a little slow especially since I had a huge dataset with many rows and columns. My first thought was that it may take a while for the necessary unsupervised methods to run completely. The first method applied was k-means clustering. This method finds cluster centers that are representative of certain regions of the data and places them in groups. Running the k-means cluster algorithm on the standardized data did not seem to produce any substantial findings. The cluster groups were on both ends of the graph and the algorithm was having a hard time of separating these two columns from one another.

Instead of placing the cluster boundary vertically down the middle, it seems that it placed it through the middle horizontally. When running k-means on the PCA transformed training set, we can see that it did a much better job at separating the points albeit not too well. The points on this training set were linear and a trend can be spotted, which is probably what helped the algorithm separate the two clusters. Applying agglomerate and DBSCAN did not work since Python kept returning an error that stated that the file was too big. I then had to work with the breast cancer dataset.

Agglomerative transformation worked well with the breast cancer dataset and was able to separate most of the points. When testing on the PCA transformed data, agglomerative clustering worked considerably better. The number of clusters I chose to fit here was five and the algorithm was capable of separating the points very well. There were still a few crossovers, but it had an easier time labeling them. The way that this process works is that the algorithm declares each point its own cluster and then combines the most similar clusters depending on the number of "n_clusters" that is stated.

Using DBSCAN, which is a method of clustering that does not require us to set the number of clusters and instead can assign them automatically. We can adjust the distance in which the algorithm classifies a cluster by changing the eps. Running DBSCAN on the cancer dataset did not seem to work for me and it seem to capture all points as one cluster. This happened for both training sets. I would have to revisit this dataset and go through it piece by piece again so that I may figure out where I might have gone wrong.

During this class I learned a lot of what it means to apply a machine learning algorithm to data and how to clean, wrangle, and apply the necessary transformation to your data depending on the type and end target that you are working with. Classification and regression data seem to be the bulk of where machine learning works its best. My classification dataset concerning the lethality of asteroids was filled with information and I think that both regression and decision trees were the optimal choices to run the algorithms on. One thing that I would have done differently was to choose a smaller dataset since Python was not capable of running agglomerative clustering and DBSCAN to my data since it was so big. I also would have chosen one where there were multiple classes to my target variable or maybe I would have worked on a regression dataset so I can have the feel for both categories and how to go about them and decide which algorithm works best. Since I have a summer all to myself I think that is exactly what I will do.