Overview

This project focused on the classification of exoplanet candidates as true exoplanets or something that produced a false reading as an exoplanet.

Data

I obtained my data from NASA's Kepler Exoplanet Archive. Kepler is a NASA mission wherein a telescope in space observes light from a star and looks for signs of an exoplanet orbiting it. When a planet passes in front of a star, it blocks some of the electromagnetic radiation from that star. For an object of interest to make it in to this dataset, there had to be some aberration in the electromagnetic radiation coming from a star observed by Kepler. Not all of these aberrations turn out to be planets though -- it could be *another* star passing between the star and Kepler's lens (ie a binary star system), it could be a brown dwarf, or instrument error.

The dataset I used was the result of Kepler observing 200,000 stars and finding 9,564 "objects of interest." Many of these objects of interest are still under consideration by NASA before they're accepted as a full exoplanet, so those were thrown out. After removing these observations and doing a bit of data cleaning, I was left with 6,809 observations -- 4,524 disconfirmed (non-planet) objects and 2,285 confirmed planet objects, giving me a ratio of about 2 negative observations for every 1 positive observation.

Tools

Python

- **Pandas**
- **Numpy**
- **scikit-learn**
- **imb-learn**

Google Docs
Google Slides
SQL (opted for challenge)
Javascript
HTML
Flask

I'm excited that I was able to build a functional Flask app for this project. I've been wanting to develop my Flask skills for a long time, but never had a good opportunity. I know Flask is an important skill for communicating between, for example, a data science department and a web development department. I feel like I added a new tool to my skill set, in that I now know how to create one of the main forms of deliverables for projects in a professional data science setting.

In building this app, I was able to get my hands dirty with HTML and Javascript. This aspect of the project showed me how much I *don't* know in these domains more than anything. I'm sure I broke plenty of rules when it comes to HTML and Javascript style guides, but in the end I got my app to render and function how I wanted it to.

Modeling

I tested many different models, including K-Nearest Neighbors, Logistic Regression, Random Forest, Linear and Sigmoidal Support Vector Machines, Gaussian and Bernoulli Naive Bayes, Stochastic Gradient Descent, Gradient Boosting, and Adaboost. I used a 5-fold cross validated GridSearch (or RandomizedSearch when I was tuning a large number of hyperparameters) with recall as my primary scoring metric. I chose recall as my scoring metric because it measures the true positive rate (aka the detection rate) -- I wanted my model to miss as few true exoplanets as possible.

Random Forest, Gradient Boosting, and Adaboost all did quite well. I could have gone with either Random Forest or Gradient Boost, but chose Random Forest for its interpretability.

Aside from stratified sampling, I tried many different undersampling and oversampling methods. Using the imbalanced learn library, I used SMOTE to upsample minority classes, TomekLink to undersample majority classes, and EditedNearestNeighbors to remove data that don't make sense when compared to nearby neighbors. I also used different combinations of these in which I used SMOTE to synthetically upsample minority classes and *then* apply TomekLink or EditedNearestNeighbors. In the end, vanilla SMOTE with no combination was the best result, increasing my cross validated recall by about .02. It was a meager gain for a lot of work, but I learned a lot about different sampling methods.

I reduced the complexity of my model by using mean decrease Gini impurity feature selection. This built-in scikit-learn method gives me the relative proportion information gained on tree splits for each feature. I was able to identify features that contributed very little to my model and drop them from my final model's features. This did not lower my recall, but it did reduce the runtime and complexity of my model significantly (I was able to drop 6 features using this method).

Conclusions

I'm very happy with how my model performed. My model's recall is .91 and precision is .92 on test data. I was very surprised by how well my model did, and considered that I might have a leaky variable, that I was making an error in cross validation. After triple checking everything

and checking the internet for other teams' results on this problem, I decided that this is a problem space that machine learning has a lot of potential in.

Currently NASA uses light curve analysis to classify exoplanet candidates. I believe my model could be used as a way to filter for candidates that are probable exoplanets, then apply light curve analysis to those candidates that are probable exoplanets, significantly reducing the number of candidates that have light curve analysis applied to them.

My model is readily available through a small local Flask app. This would allow a new set of exoplanet candidates to easily be run through my model to get predictions on-demand.

Potential Improvements

I have a bit of domain expertise in astronomy, but I found myself slightly under-equipped to handle the meaning of certain features. I wasn't able to do any feature engineering in my project's current form, and I think that's an obvious area for improvement. Choosing such a science heavy topic turned out to be difficult in this regard, and I think I'd like to focus on a business problem in the future.