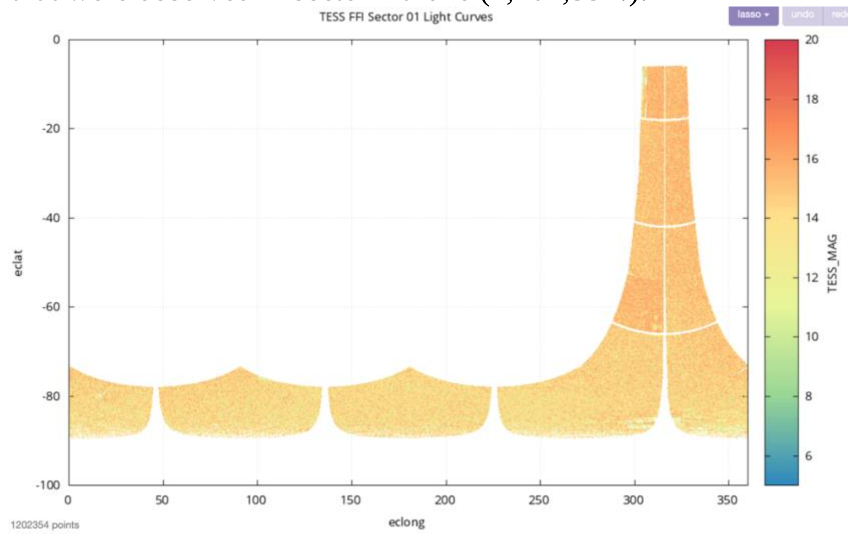
Jadyn Waggoner
CMSE 492 – Final Project Written Report

Background and Motivation:

The goal of this project is to accurately identify and catalog classical variable stars using data from the TESS primary mission and machine learning algorithms. Due to the sheer number of variable stars observed by TESS, using the data for projects on variable stars is very difficult, because data for individual star types is needed. This means that the researcher will have to manually go through millions of targets in order to find the ones that apply for their study. Utilizing the results from this project, variable stars from the TESS missions will be roughly categorized as the data becomes available. Having these categorized datasets will be useful for any scientist looking to study variable stars and quickly find the necessary data. This problem could also be solved by visually sorting the stars based on their light curves, but with tens of millions of target stars, this would be almost an impossible task. Machine learning is a much better approach because, although it is not perfect, it is much quicker at finding patterns and sorting the variable stars based on their type. There are 26 sectors, but this plot shows the large number of stars that were observed in sector 1 alone (1,202,354!):



ML task and objective:

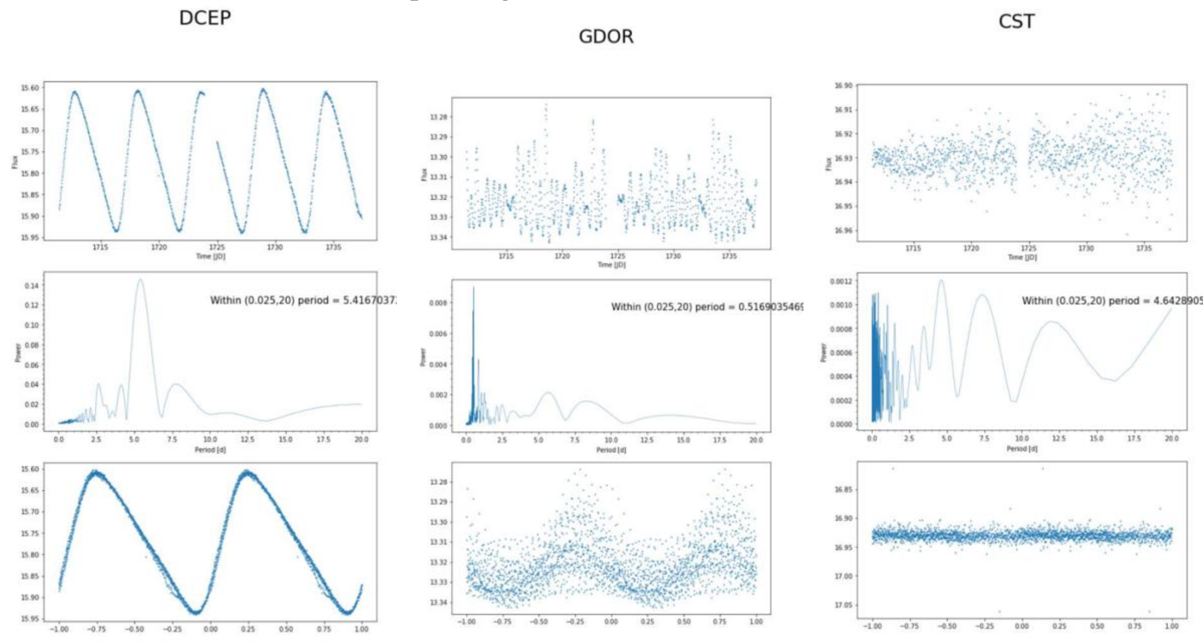
This is a multiclass classification project with 8 classes and 9 features. The classes are different types of variable stars: RR Lyraes, classical Cepheids, γ Doradus, δ Scutis, dontact eclipsing binaries, detached eclipsing binaries, non-variable/constant, and miscellaneous. The features are Jstet (J-band light curve variability), RMS_60m (root mean square deviation of flux residuals in 60-minute bins), Teff (effective temperature), w1mag-w4mag (magnitude difference between WISE filters W1 and W4), Vmag-Kmag (magnitude difference between V-band and K-band), LS_Period (period found using Lomb-Scargle Periodogram), LS_SNR (signal to noise ratio calculated from LS_Period), BLS_Period (period found using box least squares algorithm), and BLS_SDE (signal detection efficiency of BLS algorithm). I am testing the data on the KNeighborsClassifier, MLPClassifier, and RandomForestClassifier machine learning algorithms so I expect this to be a supervised learning task, but I will also be testing KMeans as an unsupervised learning task to see if it works better. Because of the messiness and noise of the data, I am going to consider metrics 0.9 and above to be acceptable.

Metrics:

For my supervised models (KNeighborsClassifier, MLPClassifier, RandomForestClassifier), I will me using accuracy score, precision, recall, and F1-score to evaluate the performance. I will know how well my model is doing because as these values approach 1, it means the model is doing a better job at classifying the star types. For my unsupervised model, KMeans, I will use silhouette score to evaluate the performance because KMeans works differently than the supervised learning models. Rather than comparing the results to preexisting data, KMeans finds clusters and then calculates the silhouette score based on how well each cluster is distinctly separated from each other.

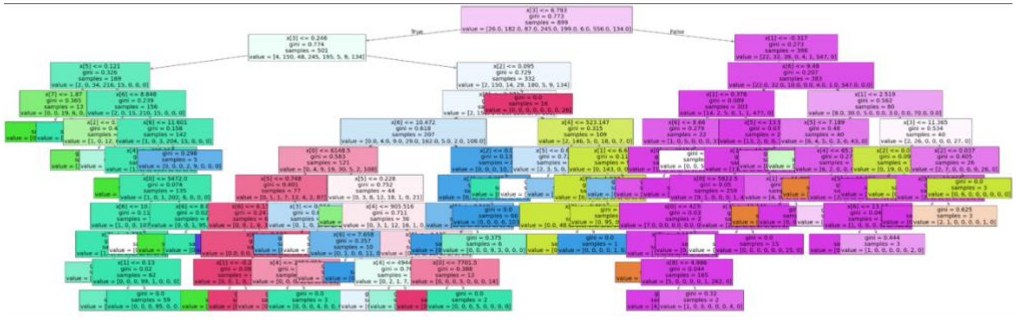
Initial and Exploratory Data Analysis:

The data is coming from this TESS Full Frame Image Portal: https://filtergraph.com/tess_ffi/camera-2-2-simulated-ffis. In order to find the type for the stars that have already been categorized, I had to upload the coordinates for each star into this website https://vizier.cds.unistra.fr/viz-bin/VizieR-3?-source=B/vsx/vsx. The data was very messy and not well categorized, I had to manually go through and clean the data to make it suitable for machine learning. I created a GUI using Tkinter which allowed me to more easily and quickly sort through all the stars I was going to use and be able to save them into their corresponding labelled files. There were a lot of stars that were incorrectly labelled, which would have messed up my machine learning algorithms. Here are some examples of the images I had pop up in the GUI so I could press the 'y' or 'n' key depending on whether it was the correct label or not.



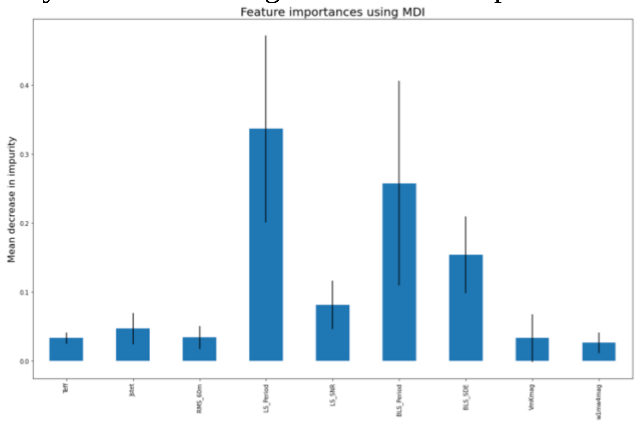
Models:

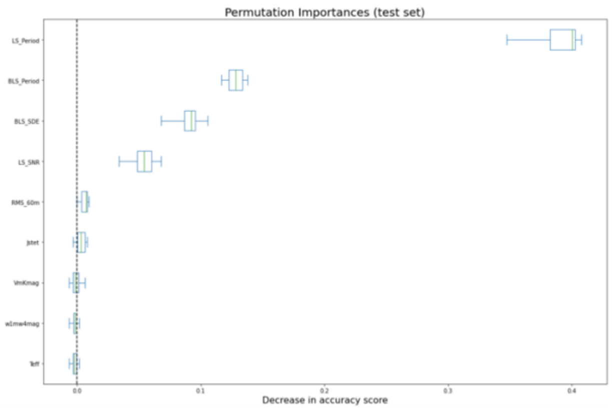
The machine learning models I will be comparing are KNeighborsClassifier, MLPClassifier, RandomForestClassifier, and KMeans. The KNeighborsClassifier classifies data points based on how close they are to their 'K nearest neighbors' on the graph. MLPClassifier stands for Multi-layer Perceptron classifier, and it is a neural network that uses neurons to receive and output data in order to classify it. The RandomForestClassifier uses many decision trees to classify the datapoints and reduce overfitting. KMeans classifier is unsupervised and uses clustering and a datapoint’s distance from cluster centers to classify the datapoints. Since each of these models classify the data in different ways, some of them may work better for this specific dataset compared to the others, and I will not know until I test each of them. Example of a random forest tree from my project, the leaves start to block each other from the amount of them, making it a bit hard to read:



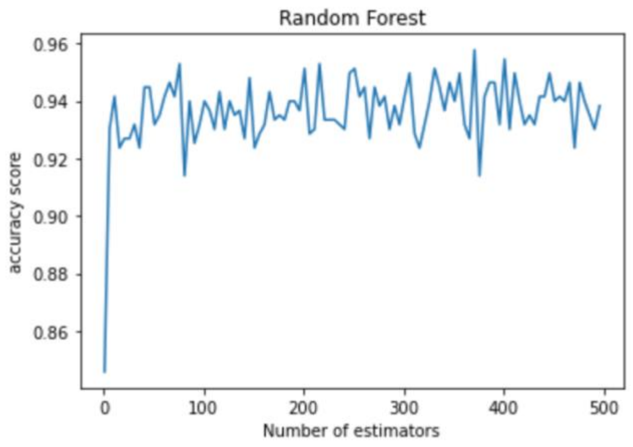
Training Methodology

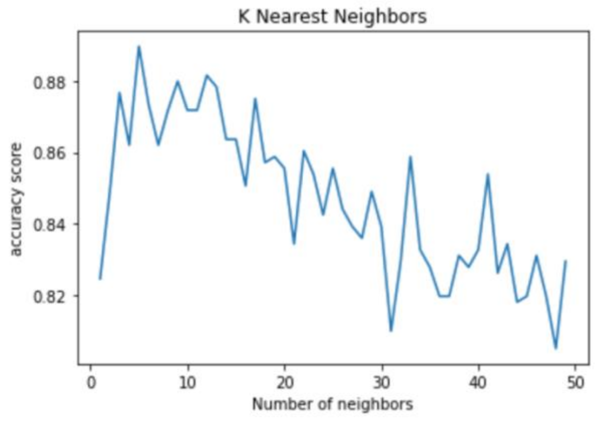
For each of the supervised learning algorithms, For the train-test split I used a 0.7 training set size and a 0.3 testing set size, to make sure that there is a good amount of data leftover after training to get useable results after testing, but making sure that there’s enough training data to properly train the model. For KNeighborsClassifier, I tuned the hyperparameter n_neighbors. The hyperparameters tuned for MLPClassifier are alpha, layer_size, and max_iter. For RandomForestClassifier, I tuned n_estimators, min_samples_split, and max_depth. For KMeans I tuned n_clusters. In order to prevent overfitting and underfitting, I used feature reduction through my functions “bestcols” and “importance”. My “bestcols” function created a model using every combination of functions, and outputs which columns cause the highest accuracy score. This prevents overfitting because it gets rid of the unimportant features that are noisy and can cause the model to fit to the noise rather than the actual trend. My “importance” function plots the mean decrease in impurity (MDI) and plots the how much the accuracy score is affected for each of the features. Using these plots, I can find the features that are most important because they will have the highest scores. The plots are here:





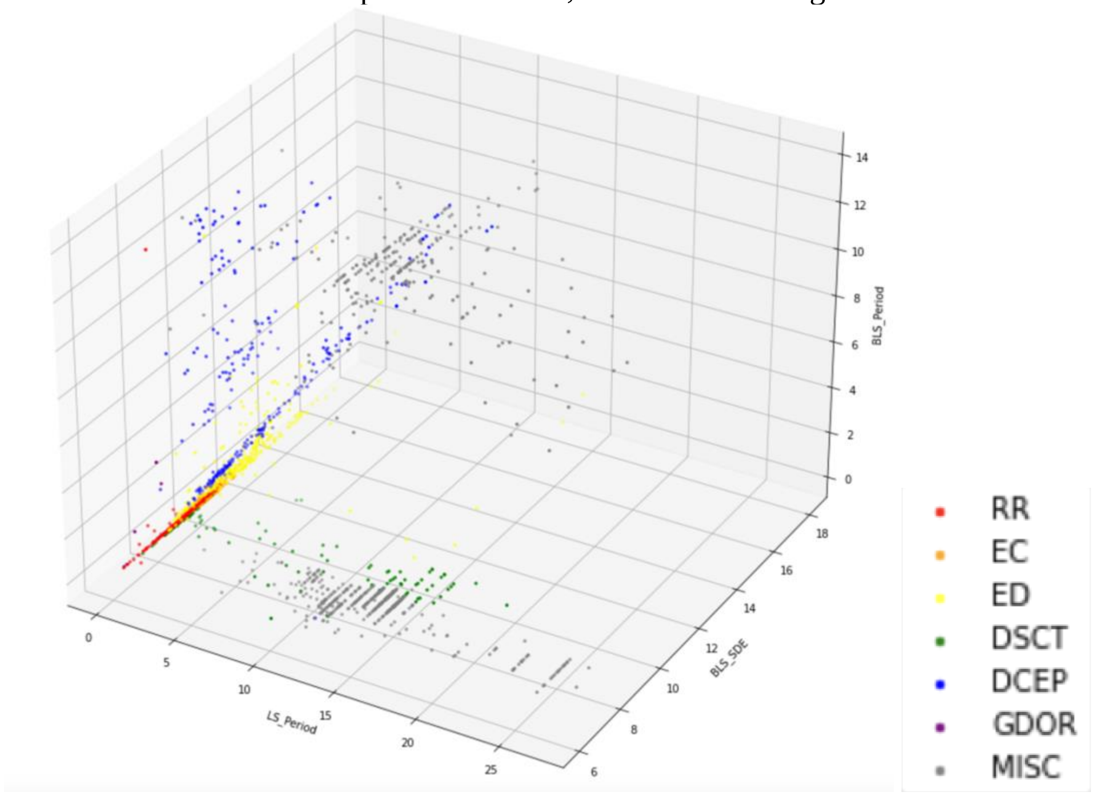
And here are some examples of how I found the best values for the hyperparameters:





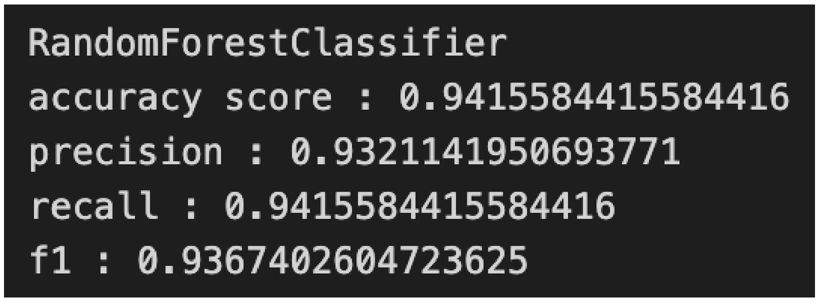
Results and Model Comparison:

For my project, all the algorithms took relatively the same amount of time (1-3 minutes) so that did not affect my choice very much. The main factors I looked into for choosing the best algorithm was the metrics like accuracy score, precision, silhouette score, etc. and the resistance to overfitting. The algorithms used for my project in order from best to worst performing are RandomForest, Kneighbors, MLP, and KMeans. The accuracy score, precision, recall, and F1-score were all above 0.93 for RandomForest, which is above my 0.9 cutoff for what I planned to consider a good model. None of the others were consistently above 0.9 (Sometimes Kneighbors accuracy score would randomly get up to 0.91, but not consistently). KMeans did not perform very well, but I didn't have very high hopes for it because of how the clusters are not very well defined even in the most important columns, shown in this image here.



Conclusion:

In conclusion, the best algorithm for my task of categorizing variable stars is the Random Forest Classifier. My algorithm achieved over 0.93 for all metrics, accuracy score, precision, recall, and F1-score, which is above my 0.9 cutoff that I stated before for what I planned to consider a good model. Some things that went wrong was that the categorized TESS data was very messy and a lot of the stars were labelled incorrectly. I had to go through them by hand to ensure they were useable for machine learning, but I did not have much time to get a lot of datapoints for the training and testing sets. In the future, to solve this issue I will spend more time cleaning more data for the training and testing sets, to make the algorithm stronger and more dependable. Another issue I ran into was that the systematics and noise in the data for each TESS sector was slightly different. To solve this for the future, I could find a way to subtract out any predictable noise that is in all the light curves/datasets for each TESS sector. Despite these obstacles, my project was successful in being able to predict variable star type with over 90% accuracy.



```
RandomForestClassifier
accuracy score : 0.9415584415584416
precision : 0.9321141950693771
recall : 0.9415584415584416
f1 : 0.9367402604723625
```