Ariel Harris

COMP 4448

Final Project Write Up

#### Executive Summary

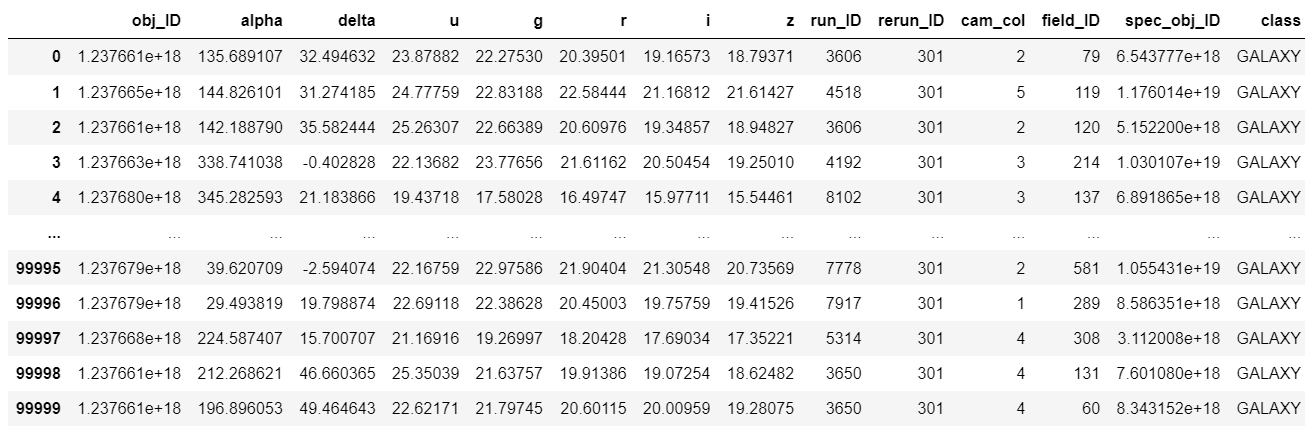
The purpose of this project is to find which of several classification models performs best on a multi-class data set, such as the stellar classification set. Specifically, Decision Tree, Random Forest, Gaussian Naive Bayes, and SVM models are all constructed and compared based on their accuracy and precision. The findings showed that a Random Forest classifier had the highest accuracy while a Gaussian Naive Bayes had the best and most even spread of precision scores. The Gaussian Naive Bayes also improved with hyperparameter tuning of the var\_smoothing parameter. This indicates that Random Forest and Gaussian Naive Bayes are the best performing mutli-class classifiers for the stellar classification dataset.

#### Research Question

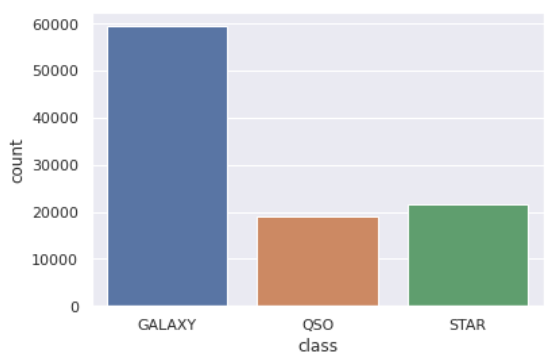
Which of several different multi-class classification models, such as random forest or naive bayes, performs best when classifying stars based on special characteristics?

#### The Data

This dataset, the stellar classification set, which is found on the Kaggle website, is used to classify stars into three different types. These types are: star, galaxy, and quasar object (a highly luminescent object near a black hole). Each data entry is described by 17 different features, 8 of which are IDs and the remaining 9 are numerical values. These represent the values of the light which filters through on the photometric spectrum. Examples would be red, infrared, ultraviolet, etc. Below is an excerpt of the data as it was provided.

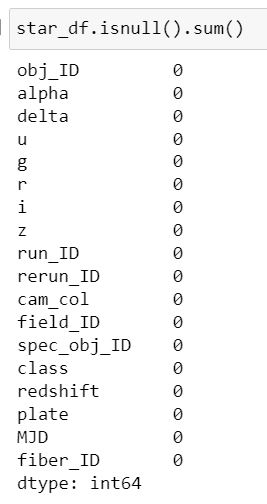
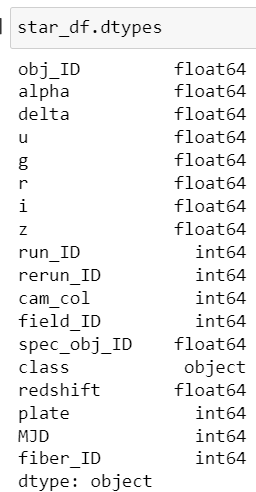


It is a larger dataset of 100,000 points, however this data is not well balanced. There are significantly more galaxy types compared to the other two classes, as is evident in the bar plot plotted below. This can lead to the model’s predicting well for galaxy but underperforming for the other two classes.

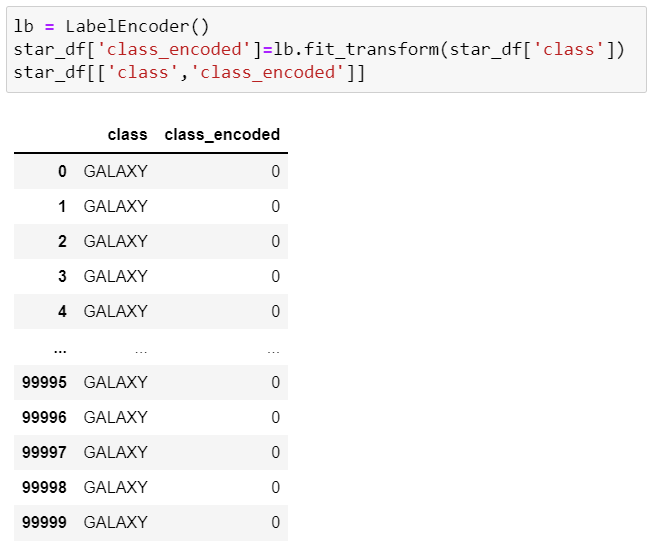


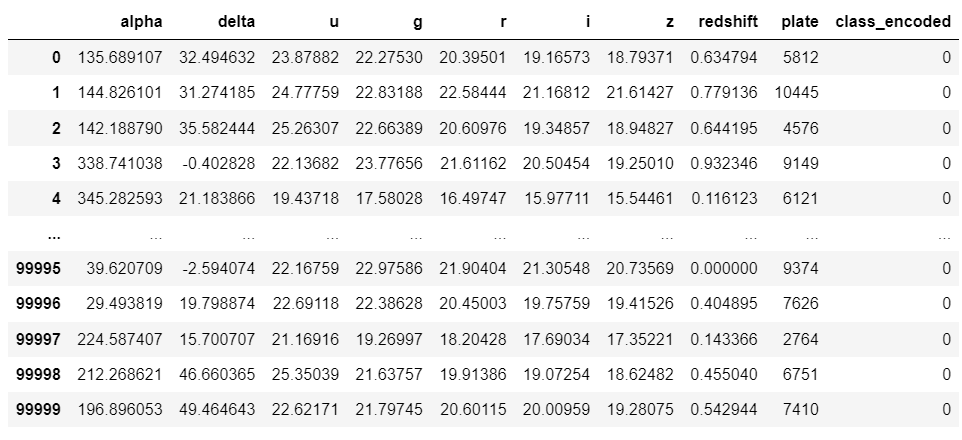
#### Data Preprocessing

The initial step for processing the data was to examine it for any missing values or incorrect data types. Fortunately, this data set was not missing any data, nor did it have any incorrect types, as can be seen in the code snippets below. As mentioned before, this is a largely imbalance dataset, however instead of resampling I wanted to see how each model I test performs with the imbalance, so the samples are left as is.

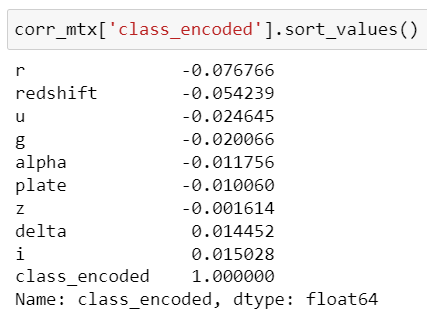


After ensuring that the basics of the data preprocessing are completed, I then removed the unnecessary ID features and encoded the class labels so they are numerical categories rather than name values. To do so I used the LabelEncoder object from sklearn. Below is a code snippet of the encoding and the same excerpt of the data set as above, but after cleaning.

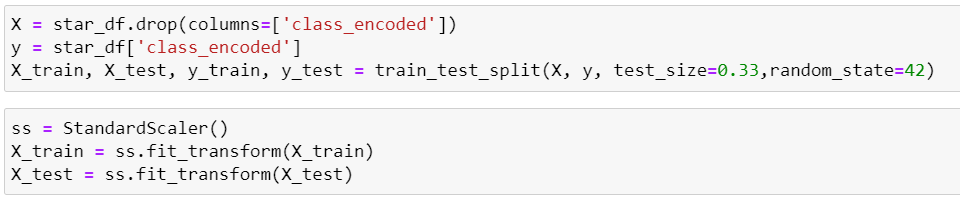




Before moving into any train-test splitting of the data, I performed some final exploratory analysis. What I wanted to confirm here was if any of the features which remained after removing the IDs were obsolete. To do this, a correlation matrix was created and a heatmap was plotted. As can be seen in the below graph, the row of values which would correlate to the class\_encoded is all of the same darker color. Also, the values in the code snippet are all near each other as well as near 0. This shows that all the remaining features are equally important, as there is no feature more or less correlated than another.

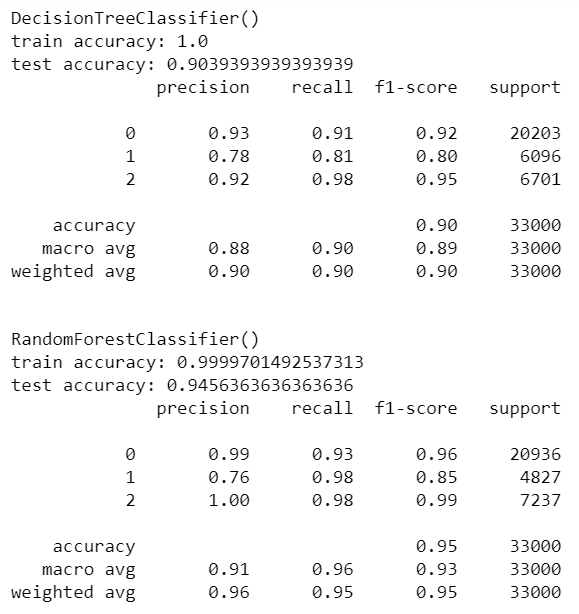
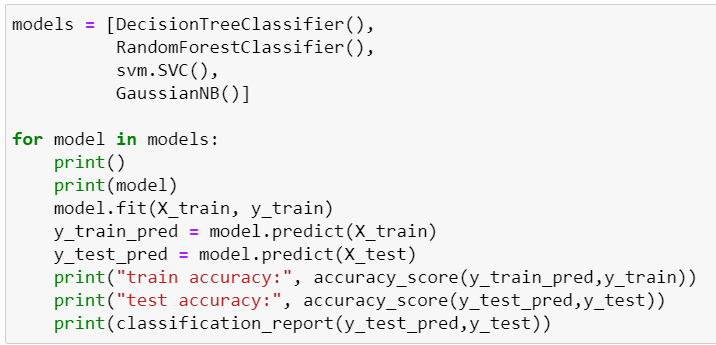


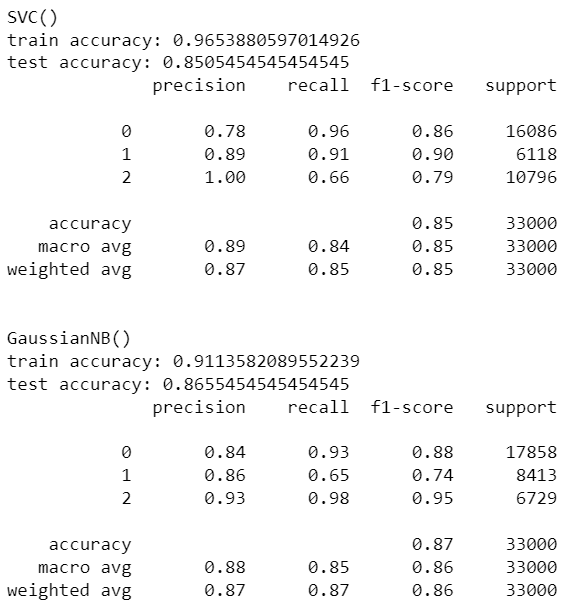
The last steps in the preprocessing is to split and scale the data in preparation for being fed into the models. Not all of the models which will be assessed required scaling, but for some of them it was a requirement, so I applied the StandardScaler from the sklearn preprocessing library. It would not negatively affect the models that do not require scaling, so it was best to apply to all for a fair assessment. To ensure that the scaler was only applied to the features and not to the target class, I applied it after doing the train-test split. Since this was a larger dataset, it wasn’t necessary to be overly careful with the split size to ensure there was enough data to successfully train on, so I did a 0.33 test split leaving 67,000 training points and 33,000 test points. Below is the code demonstrating the split and scaling.



#### Model Selection and Tuning

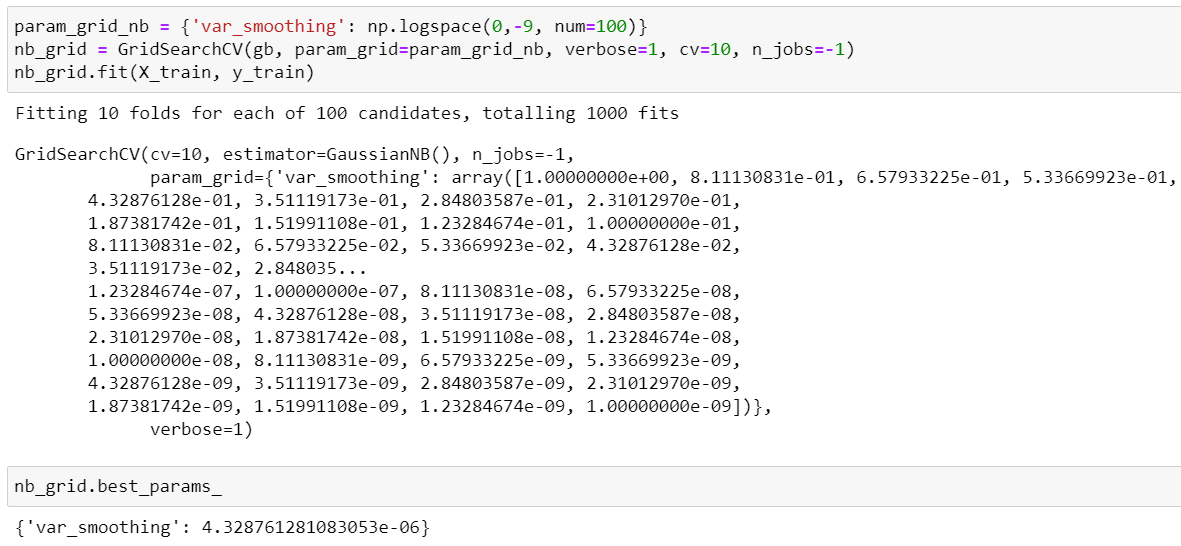
Since the purpose of this project is to determine which of several classification models is best for predicting a multi-class data set, multiple models needed to be built, fitted, and investigated. In order to do this with the most efficiency, a for-loop was used and four different models (Random Forest classifier had the highest accuracy while a Gaussian Naive Bayes) were constructed, fitted, and analized. In order to assess which of the models is considered best, the training and test accuracies are found and the classification report is printed. This provides insight into the prediction abilities of the model and any instance of overfitting that may occur. Below is the code and the resulting outputs.





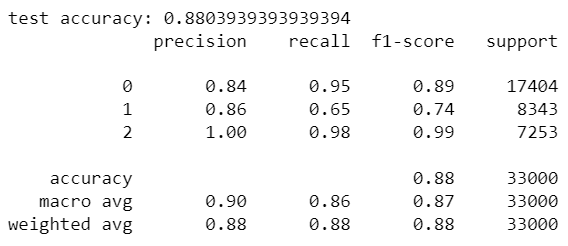
Of these models, Random Forest and Decision Tree had the highest accuracy, however the Decision Tree model showed evidence of overfitting, as it had a perfect score for the training accuracy and has the greatest difference between training and testing accuracy. Focusing on avoiding overfitting, Gaussian Naive Bayes has the closest train and test accuracy, suggesting the model would perform well on any version of the data and would be the best fit. Even though the Naive Bayes model does not have the highest accuracy, I decided that this would be the model to focus on. My reasoning for this is the spread of the precision scores for each class. It has the most even scoring for every class, whereas while Random Forest was able to predict the first and last class extremely well, the middle class was not well classified. Additionally, Gaussian Naive Bayes did not seem affected by the imbalanced data. There were far more of the galaxy class (encoded to 0) which would imply that the models would over perform when classifying that class just as Random Forest and Decision Tree did, but this model actually underperforms on that class, suggesting it is not affected by the imbalance. For this specific situation, a classifier which can perform well on each class is preferable over a classifier which performs excellently on only some of the classes.

So, the Gaussian Naive Bayes is selected to perform hyperparameter tuning. There is only one parameter to tune for this model: var\_smoothing. This parameter smoothes the curve of the model as it fits to the data. The smoothing of this curve broadens the reach of the model so more outlying points are included in the construction. To tune this parameter GridSearchCV from the sklearn model selection library was used, below is the code demonstrating the tuning and the best parameter results.



#### Model Evaluation

After performing the hyperparameter tuning, I assigned the resulting best estimator to be our final model and evaluated it on the test set. Below is the resulting accuracy score and classification report of the final model.



Running the model with the tuned hyperparameter increased the accuracy to 0.88 as well as increased the precision of the last class to being, practically, perfect.

#### Conclusions

To conclude, I found that for this multi-class classification dataset, a tuned Gaussian Naive Bayes was the best possible model. This was due to the lack of evidence for overfitting as well as the spread of precision scores suggesting it can more evenly predict across all possible classes. Though this is the conclusion I have come to, there are more steps that could be performed to improve on the model and outcomes. Firstly, the Random Forest model could have also been tuned and possibly improved to cover more of the data. Additionally, the classes for this data are greatly imbalanced, this means that there is not as much data to provide accurate predictions for each possible class. Improving on this imbalance by performing a resampling could also improve the models’ predictions. However, until these improvements could be implemented the best performing algorithm for a multi-class dataset, such as the stellar classification set, is a tuned Gaussian Naive Bayes.