Sanjana Mishra and Gerri Fox
Artificial Intelligence CS4100
June 25, 2021

Star Classification

Introduction:

The problem we decided to tackle for our final project was classifying stars based on data provided by NASA. The data in its original form had some string formatted data (for example, the color of the star was given as "yellow-white" or "blue"), but we were able to take that data and convert it into numbers through simple 1:1 matching in regards to feature engineering. This way, the data we were working with was completely numbers based, making model generation and analysis simpler. After converting our dataset to be completely numerical, we applied MinMaxScaler() to normalize the data. We used a combination of 3 ML models - KNN, Logistic Regression, and AdaBoost - in order to get a well rounded result as to which model would provide us with the highest accuracy. The model we implemented by hand was KNN and we implemented Logistic Regression and AdaBoost. One issue we faced was converting the string data into a numerical form, but we solved this by deciding to match the names of the colors with numerical values. After we did this, we realized we'd ended up with a secondary, tangential problem that we needed to solve - color and temperature are pretty correlated, which meant that we ran the risk of messing up our data through our feature engineering process. We only noticed this relationship when we realized our accuracy could have been better and began to play around with the features. To solve this issue, we applied PCA to our dataset in order to ensure the validity of our dataset and make sure that some features weren't overcounted, and our accuracy improved! We chose to use 5 neighbors based on trial and error in order to get the best results.

Analysis of Results:

For all of our models, we used the test_train split package with the stratify parameter equal outcome (which was our target column). Additionally, we had our test_size parameter equal 0.25. Our hand written KNN model with cross validation performed with an accuracy of 0.93333, while the "out of the box" implementation also performed with the same accuracy. This is likely due to the fact that our dataset was so small. Additionally, our AdaBoost results were largely positive on the testing set, with our highest accuracy being 0.95 when there were 50 base learners. Something we noticed was that our AdaBoost model was severely overfitting on the training data, and again, this was largely due to the fact that our dataset could have been larger. Lastly, we implemented Logistic Regression, and our accuracy was around 0.96 for both the training and testing set. Based on our results, Logistic Regression performed the best, with

AdaBoost next and KNN last. We recognize that these results are largely skewed due to the size of our dataset, and if we were to do the project again, would emphasize looking for a larger one.

What We Learned:

One thing we learned from the process of training our classifiers was about the test_train_split method, specifically the random_state and stratify parameters. At first, we set our random state to 3000, literally just choosing a random seed, however, when this was the seed for our test_train_split, our accuracy for classifying our test data was perfect. We thought this was very interesting that we managed to choose a completely random seed that somehow ended up creating a perfect classifier, at least on this data sample, so ultimately we ended up with just leaving this parameter as the default and going completely random each time. When investigating this, we also learned about the stratify parameter also on the test_train_split method. We thought this would be beneficial for our model to include as it would ensure that all star types are equally represented in our training data so as to not let the classifier mistake one type for being more common than another. But, as a result of learning more about these parameters for test_train_split, we ultimately came to the conclusion that our dataset may just be too small and is thus naturally resulting in overfitting. From this, we learned about the importance of having a large dataset. Typically, we understand that K-nearest neighbors does not usually produce the best results, however, since our dataset is miniscule, knn proved to not be that inaccurate. But, we realize that this is not a realistic representation of real world classification issues and thus we learned how important it is to have a large, inclusive and representative dataset in order to accurately and efficiently train any machine learning models. Another thing we learned was the importance of creating a good experiment design. For example, our dataset included String data such as star colors and spectral classes which do not register with machine learning techniques. So, we conducted feature engineering and designed proper data to be applicable to machine learning models and AI techniques. Lastly, we took an in depth look into Adaboost. Specifically, we chose to "boost" a Decision Tree classifier. One of the parameters we learned about was the "average" parameter. By default it is set to "binary," however, since our data is not true/false we had to change this parameter to "macro." Thus, we initialized a basic Decision Tree classifier as our base estimator and by using AdaBoost, we learned that boosting really works to prevent overfitting. As I mentioned before, we believe since our dataset is so small, that naturally allowed KNN and Logistic Regression to be more effective than they realistically would, thus although in our model our AdaBoost classifier has the lowest accuracy, our dataset is not the most representative of the real world, a valid comparison if you will may even be the simple "Box World," but, it is the most realistic and will be the most applicable to real world classification problems.

Appendix/Step by Step Instructions:

1. Download the zipped file provided on Canvas (the zipped file should contain both the Jupyter Notebook and the CSV that we worked on)
2. Once downloaded, open the zip file and ensure that the CSV and Notebook are in the same folder in Jupyter Notebook
3. Open Jupyter Notebook and run all the cells (there is no random_state set, so the accuracy may vary when run)