Stellar Classification - DL Final

Yingying Deng, Zac Rios

December 15, 2023

1 Introduction

In the field of astronomy, stellar classification is a fundamental process for categorizing stars according to their distinctive spectral characteristics. This classification system is essential not only for stars but also for galaxies and quasars. The early efforts to catalog stars and map their positions in the night sky played a crucial role in realizing that these stars collectively form our Milky Way galaxy. Furthermore, as astronomical observations progressed and it was established that Andromeda is a distinct galaxy separate from our own, the exploration of numerous other galaxies became possible with the advent of increasingly powerful telescopes. In this project, we will aim to classify stars, galaxies and quasars based on their spectral characteristics. We use modern tree-based machine learning methods, and compare those to neural networks with hyperparameters optimized by grid search and random search. This problem has been explored quite a bit, but there has generally been a lack of comparison between the performances of optimized neural networks and tree-based methods.

2 Data Visualization

The dataset we are working with contains 100,000 observations is taken from Kaggle [1]. The observations are all stellar objects originally mapped by the Sloan Digital Sky Survey.It contains 17 variable columns and 1 class column. Our response variable is "class," which is a factor with three levels known as "GALAXY," "QSO" (quasar object) and "STAR". The class distribution is as follows: 59445 are galaxies, 21594 are stars, and 18961 are quasars. The pie chart in Figure 1 shows the class distribution.

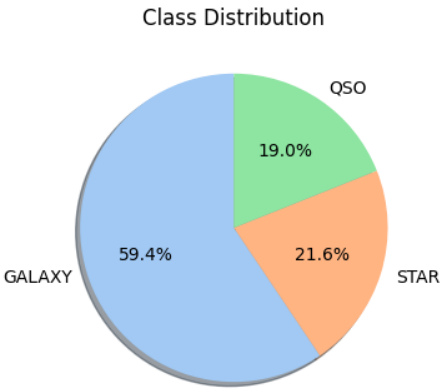

Figure 1: Stellar Objects Class Distribution

To visually depict the distributions inherent in designated columns within the dataset, we employed a grid-based approach to generate Kernel Density Estimation (KDE) plots specific to the identified columns[2]. These distributions were stratified according to the categorical variable 'class.' The resultant KDE plots stand as pivotal instruments for conducting a rigorous comparative analysis, facilitating the examination of distributions across a spectrum of classes and the entirety of the dataset. Figure 2 illustrates the KDE graph. Interpreting the graph allows for the discernment of features critical to class differentiation. Specifically, the variable "redshift" exhibits disparate distribution patterns across classes, rendering it a salient and informative variable for our analytical purposes. In contrast, the variable "object" manifests an invariant distribution pattern across classes, thereby indicating its limited utility for our analysis.

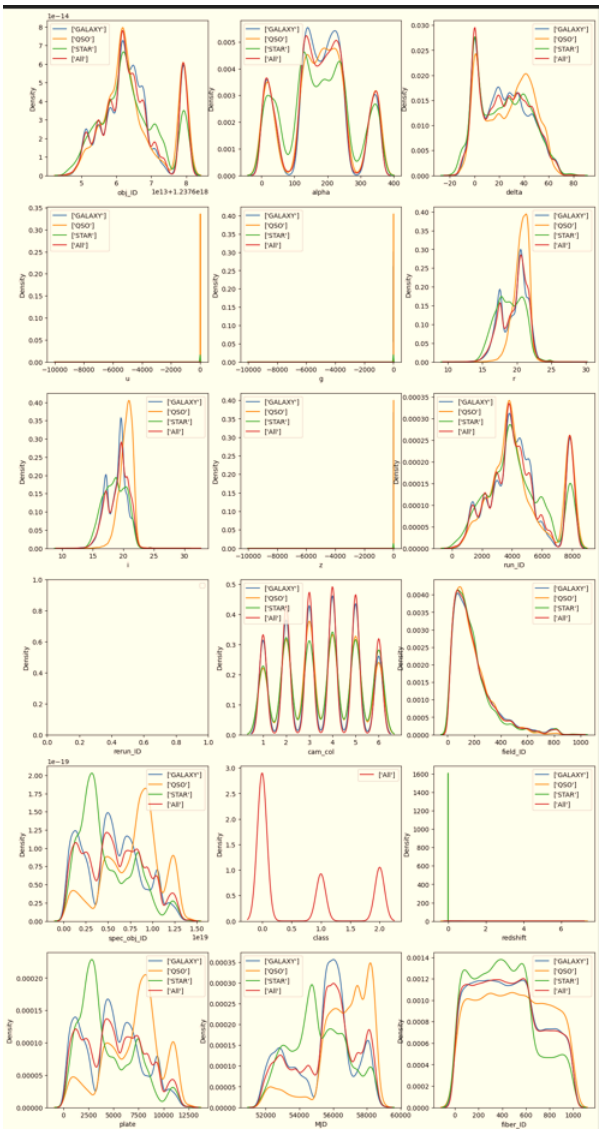

Figure 2: The KDE Graph

3 Data Analysis and Preprocessing

Within our dataset, it is noteworthy that none of the features exhibited missing or null values. Additionally, certain features contained information that pertained to instance identification or camera information. Despite their inclusion, these features are not relevant for determining the class to which an instance belongs. Consequently, these particular features were intentionally excluded from our analysis.

Feature selection was executed through the utilization of a correlation heatmap, which depicted correlation coefficients between pairs of variables (Figure 3). Variables with negligible relevance and low relationships were subsequently eliminated, resulting in the retention of 9 pertinent variables: "u," "g," "r," "i," "z," "spec_obj" "redshift," "plate," and "MJD." The correlation among the chosen variables is presented in Figure 4

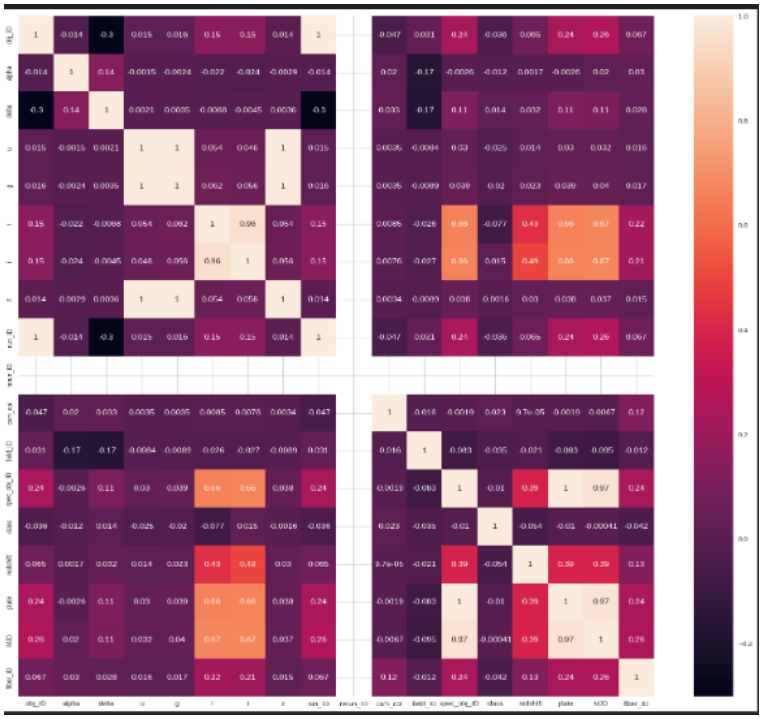

Figure 3: Feature Correlation Heatmap

Figure 4: Feature Reduced Correlation Heatmap

The "class" column underwent transformation using LabelEncoder in python, a process through which class labels were converted into integer representations, rendering them compatible with machine learning algorithms. To mitigate class imbalance within the training data, we employed the Synthetic Minority Over-sampling Technique (SMOTE). This technique involves setting a random state and specifying the number of nearest neighbors (k_neighbors). Through these parameters, SMOTE generates synthetic instances within the minority class, effectively balancing the class distribution in the training data. This resampling strategy is designed to enhance the model's sensitivity to the minority class, thereby potentially improving performance when confronted with imbalanced data, as is the case in our dataset.

The dataset has been partitioned into three distinct subsets: training, validation, and test sets. The initial division involves splitting the data into training (80%) and test (20%) sets, employing a random seed of 42. Subsequently, the training set is further divided into training (90%) and validation (10%) subsets, utilizing a random seed of 24. This sequential partitioning strategy is designed to enable the model training on one subset, fine-tuning with the validation set, and ultimately assessing performance on the independent test set. This comprehensive approach ensures a robust evaluation of the model's capabilities and generalizability.

Feature scaling is systematically applied to the training, validation, and test datasets, employing the StandardScaler from the scikit-learn library. This transformation ensures uniformity across feature dimensions by setting the mean to 0 and the variance to 1. Such standardization proves beneficial in facilitating neural network training. Subsequently, the scaled datasets are converted into PyTorch tensors, offering compatibility with PyTorch functions and enabling accelerated processing on a GPU.Moreover, this transformation extends to the target labels, which are also converted into PyTorch tensors while preserving their original datatype as long integers. This comprehensive preprocessing approach establishes a standardized and compatible input format for subsequent neural network training and evaluation procedures.

4 Models and Results

4.1 Evaluation

A confusion matrix is among the best ways to evaluate a classification models performance. It provides a summary of a model's performance through the enumeration of counts corresponding to true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions for each class. Utilizing the confusion matrix, we assess each models precision, recall, F1-score, and accuracy, metrics crucial for understanding different facets of its classification performance. Precision denotes the accuracy of positive predictions, recall quantifies the model's ability to capture all positive instances, the F1-score harmonizes precision and recall, while accuracy provides an overall measure of correctness. The formulas are as follows:

$$precision = \frac{TP}{TP + TN}$$

$$recall = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2}{(\frac{1}{precision}) + (\frac{1}{recall})}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

For comparative analyses among models, we employed a normalized confusion matrix. Normalization allows for a proportional assessment of model performance, accounting for potential class imbalances. This approach ensures a more nuanced evaluation, particularly when the distribution of classes may vary, offering a comprehensive and equitable basis for model comparison.

4.2 Baselines

4.2.1 Random Forest

Random Forest is a widely adopted supervised learning algorithm capable of handling both classification and regression tasks. As an ensemble algorithm, it comprises multiple decision trees. The algorithm makes classification decisions for an instance based on the majority class chosen by the constituent decision trees. It entails various hyperparameters, but in this implementation, specific parameters were tuned for optimization. These hyperparameter values are shown in Table 1, with the methodology of tuning explored in [3].

	Value
n_estimators	30
max_depth	14
min_samples_split	2
min_samples_leaf	2
max_features	4

Table 1: Random Forest Hyperparameters

The results of the random forest are shown in Figures 5 and 6.

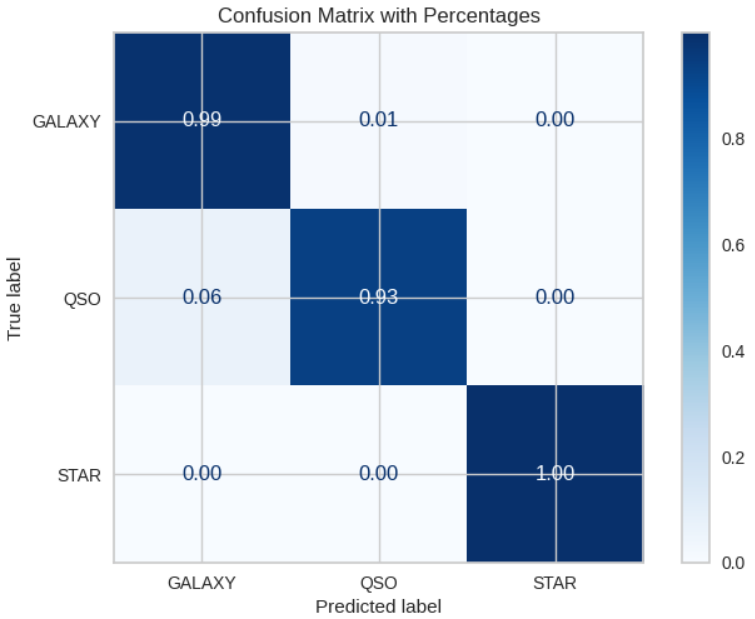

Figure 5: Random Forest Normalized Confusion Matrix

	precision	recall	f1-score	support
0	0.98	0.99	0.98	11851
1	0.96	0.93	0.95	3835
2	1.00	1.00	1.00	4314
accuracy			0.98	20000
macro avg	0.98	0.97	0.98	20000
weighted avg	0.98	0.98	0.98	20000

Figure 6: Random Forest Results

The Random Forest model had an accuracy of 98%. The model correctly classified all the star instances, where both recall and precision scores were 100%. Recall and precision scores for class galaxy were 99% and 98%, respectively. For quasar instances, recall score was 93% and precision score was 96%.

4.2.2 Gradient Boosting

Gradient boosting is another highly popular tree-based method for tabular data. Gradient boosting typically generalizes extremely well, captures dependencies in data, and often provides the highest accuracy out of all ensemble methods. The hyperparameters for our GBM model were also tuned using sci-kit learn. The results for Gradient boosting are shown in Figures 7 and 8.

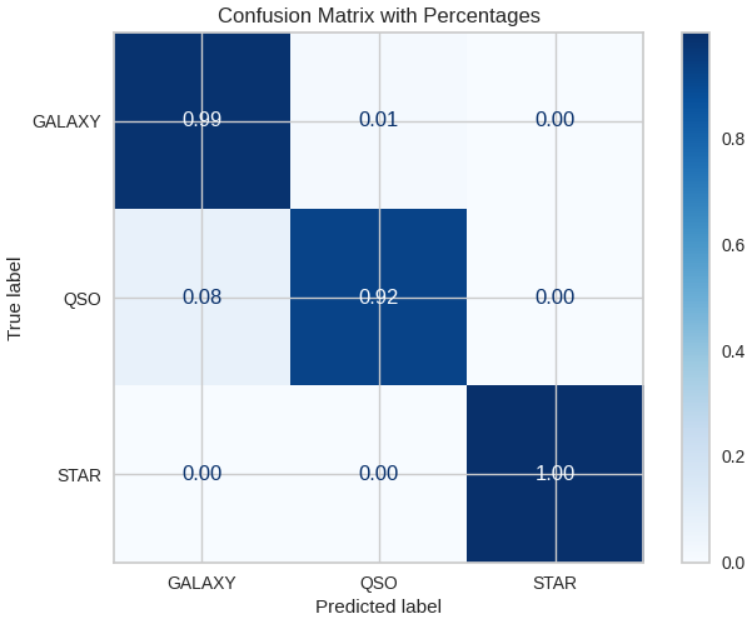

Figure 7: GBM Normalized Confusion Matrix

	precision	recall	f1-score	support
0	0.97	0.99	0.98	11851
1	0.96	0.92	0.94	3835
2	1.00	1.00	1.00	4314
accuracy			0.98	20000
macro avg	0.98	0.97	0.97	20000
weighted avg	0.98	0.98	0.98	20000

Figure 8: GBM Results

The GBM model had an accuracy of 97.7%. The model correctly classified all the star instances, where both recall and precision scores were 100%. Recall and precision scores for class galaxy were 99% and 97%, respectively. For quasar instances, recall score was 92% and precision score was 96%.

4.2.3 Multinomial Logistic Regression

While we aren't particularly interested in Logistic Regression for its accuracy or other evaluation metrics, it does give us insight into the parameters that matter, and the worst case scenario for a neural network. When we run logistic regression on our data, we get parameter estimates for redshift of 5.43 for Galaxy to QSO and -871.45 for Galaxy to Star. These are far and away our largest parameter estimates for any factor, and show that redshift alone can explain a lot of our class variation. The other important takeaway of logistic regression is its accuracy of 94.63%. Multinomial logistic regression can be thought of as the simplest possible implementation of a neural network (no hidden layers, softmax activation), so we have an important standard to compare our networks to.

4.3 Neural Networks

A multilayer neural network, often referred to as a feedforward neural network or a deep neural network, is a type of artificial neural network with multiple layers of nodes (neurons) organized in a sequential manner. Each layer is connected to the subsequent one, and the network comprises an input layer, one or more hidden layers, and an output layer.

In this implementation, we defined a Stellar Classifier class which was a custom neural network model designed for classifying stellar objects. The basic architecture is displayed in Figure 9. We used two different approaches to optimize the hidden layer sizes, the learning rate, the number of epochs, and the dropout rate. We used the Adam optimizer and CrossEntropyLoss function.

```
StellarClassifier(
  (dropout): Dropout(p=0.67, inplace=False)
  (layers): Sequential(
    (0): Linear(in_features=9, out_features=256, bias=True)
    (1): LeakyReLU(negative_slope=0.01)
    (2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Dropout(p=0.67, inplace=False)
    (4): Linear(in_features=256, out_features=256, bias=True)
    (5): LeakyReLU(negative_slope=0.01)
    (6): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): Dropout(p=0.67, inplace=False)
    (8): Linear(in_features=256, out_features=256, bias=True)
    (9): LeakyReLU(negative_slope=0.02)
    (10): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): Dropout(p=0.67, inplace=False)
    (12): Linear(in_features=256, out_features=3, bias=True)
  )
)
```

Figure 9: The Structure of the NN

4.3.1 Grid Search

In the Multilayers Neural Network, we applied Grid Search to tune the hyperparameters,the best hyperparameters are in figure 8. The model had an accuracy of 97%. It performed worse than Random Forest for three category instances. For the class star, recall and precision were 100% and 97%, respectively. For class quasar, recall score was 92%, and the precision score was 94%. For galaxy instances recall and precision scores were 97%.

```
Best Hyperparameters: {'hidden_size1': 256,
                       'hidden_size2': 256,
                       'hidden_size3': 256,
                       'num_classes': 3,
                       'learning_rate': 0.1,
                      'num_epochs': 150,
                      'dropout': 0.67}
Best Validation Accuracy: 0.9688
```

Figure 10: Grid Search Neural Network Hyperparameters

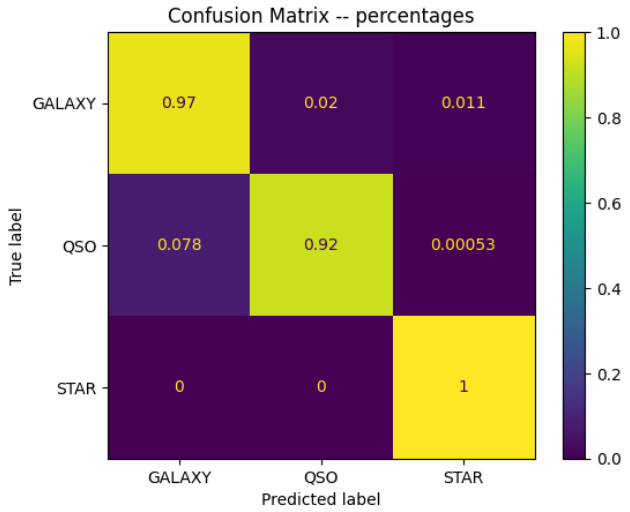

Figure 11: Confusion Matrix for Grid Search NN

	precision	recall	f1-score	support
0.0	0.97	0.97	0.97	11860
1.0	0.94	0.92	0.93	3797
2.0	0.97	1.00	0.98	4343
accuracy			0.97	20000
macro avg	0.96	0.96	0.96	20000
weighted avg	0.97	0.97	0.97	20000

Figure 12: Multilayers Neural Network Results

4.3.2 Random Search

We use random search to optimize the hyperparameters of our neural network as well. Our selected hyperparameters are listed in Table 2

Random Search					Optimal
HS1	32	128	64		128
HS2	16	64	256		256
HS3	64	32	16		64
Learning Rate	0.006	0.04	0.0001		0.0001
Number of Epochs	50	100	150		150
Dropout	0.13	0.26	0.41	0.7	0.41

Our test accuracy for this model was 96.97%. All recall and precision scores were functionally identical to the grid search case. The confusion matrix for this network is shown in Figure 13.

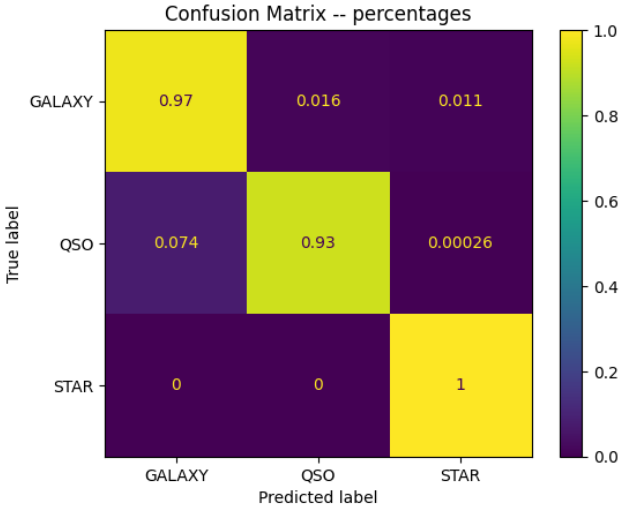

Figure 13: Confusion Matrix for Random Search NN

5 Discussion/Conclusion

Our results show about what we'd expect given the nature of our problem. Our data is simple and tabular, which lends itself well to tree-based methods. The accuracy of our neural networks never quite reaches the accuracy of both Random Forest and GBMs. Our accuracy for these tree based methods are in line with the best previous results for classification of this data (the best other attempts on Kaggle plateau around 98% as well). Our networks after HPO are able to perform better than logistic regression, demonstrating their ability to be modified to better solve most problems. Random search and grid search performed similarly, with there being no real difference between the two. This also aligns with our intuition, the hyperparameters matter very little past a certain point for this kind of data. Ultimately, all models do perform quite well, likely using the redshift of an image as the primary driver for classification. In the future, it would be worthwhile to find stellar image data to use a neural network on directly. The advantages of a neural net over tree-based methods would likely become much clearer when the structure of the data is more complicated.

6 Contributions

The contributions of each group member are listed below

- Yingying - Preprocessing, Random Forests, Grid Search Neural Network
- Zac - Gradient Boosting, Multinomial Logistic Regression, Random Search Neural Network

Both group members contributed to writing both the presentation and the report.

Bibliography

[1] Data:
https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss17
[2] Hyperparameter Tuning the Random Forest in Python using scikit-learn
https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
[3] Understanding & Interpreting Confusion Matrix in Machine Learning (Updated 2023)
https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/#:~:text=A.,positive%2C%20and%20false%20negative%20predictions