Comparison of Supervised Machine Learning Models for Stellar Classification

Artur Correia
DETI, UA
Universidade de Aveiro
Aveiro, Portugal
NMEC: 102477

Daniel Carvalho
DETI, UA
Universidade de Aveiro
Aveiro, Portugal
NMEC: 77036

***Abstract*—In this project, various supervised machine learning (ML) models were trained to classify instances from the Sloan Digital Sky Survey Data Release 17 (SDSS DR17) as either Galaxies, Stars or Quasars. The goal was to select models already tested in related work, as well as to train new models not used before on this problem, and then compare their results in terms of efficiency and accuracy, based on various performance metrics. Hyper-parameter tuning was also applied, to check whether this had any effect in bettering the efficiency of the tested models. The models trained were Logistic Regression (with or without regularization), Support Vector Machine (SVM), XG Boost, Random Forest and Neural Network. Random Forest and XG Boost achieved the better scores, with balanced accuracies of 97%, and Precision, Recall and F1-Scores of over 96%.**

***Index Terms*—Machine Learning; Classification; Stellar Classification; Supervised Learning; Logistic Regression; SVM; XG Boost; Random Forest; Neural Network; Hyper-Parameter Tuning.**

I. Introduction

The following report describes the elaboration of the first group project for Tópicos de Aprendizagem Automática, serving to contextualize the chosen problem and data set, explain the methodologies and techniques used for data processing, as well as for the training of the applied machine learning models, and to present and discuss the obtained results.

A. *Contextualization*

The dataset chosen for this project was the Stellar Classification Dataset SDSS17, published here. [1] This data set belongs to SDSS, the Sloan Digital Sky Survey funded by the Astrophysical Research Consortium (ARC), one of the most influential surveys done in the history of astronomy, that uses major multi-spectral imaging and spectroscopic surveys redshift to create some of the most detailed three-dimensional maps of the Universe ever made. [2] The activity of this project started in 2000, and since then, their research has contributed to, among others, advance the understandings of cosmological measurements of the critical early phases of cosmic history, resolve individual mapping of various galaxies, supermassive black-holes, and extra-solar giant planets, as well as mapping the clustering of galaxies and intergalactic gas in the distant universe and studying the chemical evolution of the Milky Way. [3]

These studies done by SDSS were very important not only due to the advancements in spectroscopy and imaging techniques in astronomical observations, but also to the advancement of data reduction and storage techniques, as the volume of data generated by the telescopes is enormous and unprecedented. In fact, SDSS contributed to advances in technologies such as SQL.

Most of the data obtained in the SDSS surveys is published yearly, as was the case of the dataset analyzed in this project. This dataset was released on the SDSS DR-17, the final data release of the fourth phase of the project, containing observations done in 2021. [4] In this case, the features present are related with the problem of stellar classification, that is, the classification of stars based on their spectral characteristics, with the fundamental classification being the distinction of celestial bodies in either stars, galaxies, or quasars. Among the spectral information used to do this distinction are photometric information, with the UBV photometrics system being used to classify stars according to their colors; alongside it, the analysis of redshift, the amount to which light of an object is stretched when shifted towards the red part of the spectrum, which can be used to determine how an object is moving compared to Earth, and as such, measure the distance between it and other objects. [5]

Although the definition of stars and galaxies is very well-known, the same isn't true for quasars. Quasars, or quasi-stellar radio source, are young galaxies, generally located at vast distances from Earth, towards the edge of the visible Universe. They are currently theorized to be the extremely luminous centers of infant galaxies, usually powering supermassive black holes on their center, with most large galaxies believed to have gone through this quasar phase on their early formation. [6]

B. *Related Work*

Machine learning algorithms have been very broadly applied in problems of stellar classification. In fact, we have found a few papers using this dataset for Machine Learning training, as well as other unpublished reports and studies.

Omat et al., in 2022, trained eight classification algorithms using this data set. They were decision trees, K-Nearest neighbors, multinomial logistic regression, multilayer perceptron,

a Naïve Bayes classifier, a random forest, support vector machine (SVM) and a voting classifier. Most of the studied algorithms obtained accuracy values of 95%+, with good recall, precision, and F1-score values as well, particularly on the identification of stars. [7] Qi et al., also in 2022, compared 3 models, namely decision trees, random forest and SVM. Again, values of 95%+ for accuracy, recall, precision, and F1-score were reported. [8]

A few studies were also available on the Kaggle platform where the dataset was taken from. Once again, most of the studies involved the testing of random forest and SVM, with strong values for all the performance metrics being achieved. [9], [10]

However, in most studies, no mention of hyper-parameter tuning was made, with most using default values for the hyper-parameters of their models. In fact, only Omat et al. mentioned doing hyper-parameter tuning as part of the model training approach. However, they only explained their tuning process for the decision trees algorithm, being left unclear if the same procedure was used on the training of all machine learning models. Besides this, when analyzing the data pre-processing done in these studies, we felt like changes could be made to better the model in nearly all of them – for example, some didn't eliminate similar features that might bias the model, as was the case on Qi et. al's reports; others didn't properly balance the data set used for training, given that the number of examples classified as stars in the dataset is much bigger than the number of examples classified as galaxies or quasars, as was the case for Omet et al.

Therefore, the main goal of the current project will be to validate the results for some of the machine learning models found on these papers that found the highest performance metrics (namely the random forest and SVM), as well as applying other new models that we find interesting and with potential to also give accurate results, in this case logistic regression with and without regularization, neural network, and XG Boost.

We will, however, conduct our own data analysis and pre-processing, as well as do hyper-parameter tuning for all the machine learning models, training an optimized model at the end of the process. We will compare the results obtained with each model, as well as with the models present in the related studies.

II. Exploratory Data Set Analysis and Data Pre-Processing

A. *Data Set Description*

The dataset utilized in this study comprised 100,000 observations obtained from the Sloan Digital Sky Survey (SDSS). Each observation within the dataset was described by 17 distinct feature columns and a single class column, which served the purpose of classifying observations into one of three categories: star, galaxy, or quasar. Figure 1 visually represents the distribution of the dataset, indicating 59,445 examples of galaxies, 21,594 examples of stars, and 18,691 examples of quasars.

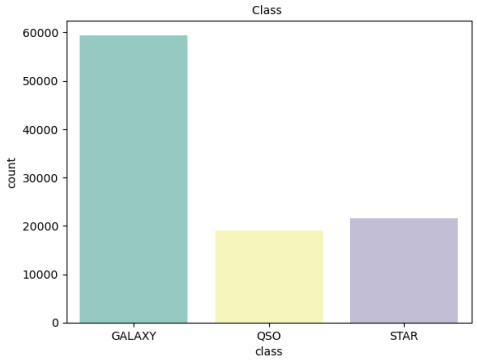


Fig. 1: Initial Class Distribution

B. *Feature Engineering*

The feature set in the dataset consisted of various columns serving different purposes. Columns such as obj_ID, run_ID, rereun_ID, cam_col, field_ID, spec_obj_ID, plate, MJD, and fiber_ID were used solely for identification. The alpha and delta columns provided information about the object's position within the celestial sphere. The columns u, g, r, I, and z were associated with the photometric system, providing measurements of the object's brightness in different wavelength bands. Lastly, the redshift column indicated the shift in wavelength due to the object's motion and served as a measure of distance. [11]

During data preprocessing, we ensured the absence of missing or null values in the feature columns. We evaluated the relevance of each feature for the model's training and excluded insignificant ones, including identification columns like obj_ID. Similarly, the alpha and delta columns indicating object positions were deemed irrelevant and eliminated. As a result, the refined feature set comprised six variables.

Recognizing the presence of class imbalance, which can introduce biases in machine learning models, a data balancing approach was deemed necessary. To address this concern, the Synthetic Minority Oversampling Technique (SMOTE) was selected for oversampling. SMOTE is an algorithm that effectively handles class imbalance by generating synthetic samples for the minority class. By identifying examples from the minority class that are proximal in the feature space, SMOTE constructs a line and generates new samples along that line. This technique mitigates the class imbalance issue, leading to a more balanced representation of the classes in the dataset. The impact of applying SMOTE on the class distribution is illustrated in Figure 2, highlighting the achieved improvement in class balance. [12]

We computed a correlation matrix (Figure 3) to evaluate the relationships between the selected six features. A correlation value of 1 indicated high similarity. After analyzing the matrix, we observed that u, g, and z features were highly correlated (correlation = 1), indicating redundancy. To avoid duplication, we decided to exclude u and z from the feature set. Consequently, the final feature selection (g, r, i, and redshift)

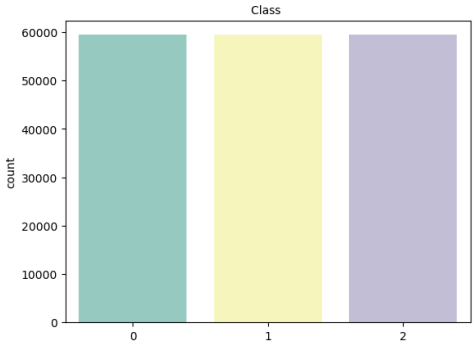


Fig. 2: Balanced Class Distribution

is presented in Table I, as these variables provide unique and valuable information for the model. [13]

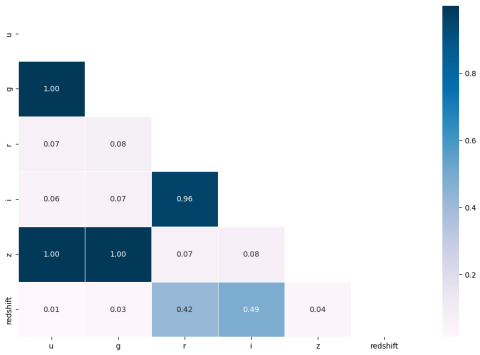


Fig. 3: Correlation Matrix

TABLE I: Final Features description

Feature	Type	Description
g	float	Green filter in the photometric system
r	float	Red filter in the photometric system
i	float	Near-infrared filter in the photometric system
redshift	float	Redshift value based on the increase in wavelength

C. *Data Analysis and Normalization*

To gain a comprehensive understanding of the features, we utilized box-and-whisker plots (Figure 4) to summarize the central tendency, variability, and skewness of their distributions. During this analysis, we identified distinct outliers that significantly affected the data visualization, for instance in the plot of the feature g, there was an outlier so distinct from the others that we could not see the box. Given the potential impact of outliers on statistical analysis and model performance, we took the necessary step to eliminate them.

Additionally, we utilized a scatterplot matrix (Figure 5) to visually examine the relationships between pairs of features. The scatterplot matrix provides a comprehensive overview of the pairwise interactions among the selected features. Each plot in the matrix represents the relationship between two features, with the diagonal squares displaying kernel density estimation (KDE) plots, illustrating the distribution of each

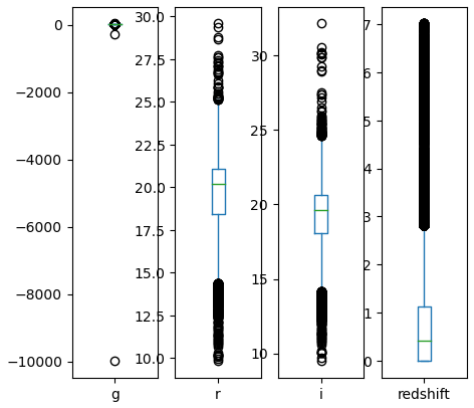


Fig. 4: Box and Whisker plot for each feature

feature individually. This analysis further confirmed the presence of outliers, reinforcing our decision to address this issue.

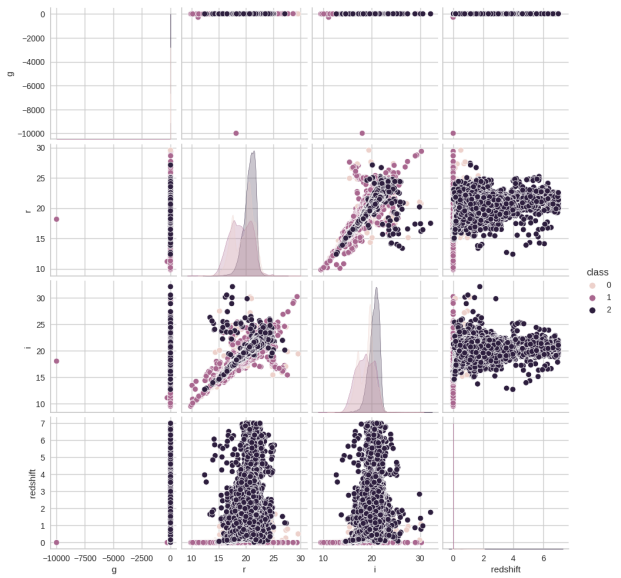


Fig. 5: Scatter plot matrix before removing outliers

To address the issue of outliers, we employed the Interquartile Range (IQR) method. The IQR is a statistical measure that represents the range between the first quartile (Q1) and the third quartile (Q3) of a dataset. Any data points falling below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR are considered outliers. [14]

In our analysis, we applied the IQR method to identify and remove outliers from the dataset. By calculating the IQR for each feature, we determined the range within which the majority of the data points lie. Any values outside this range were considered outliers and were subsequently eliminated from the dataset.

Following the removal of outliers using the IQR method, a new Scatterplot Matrix (Figure 6) was generated. This revised matrix provided a clearer and more informative visualization of the data, as it allowed us to examine the relationships

between features without the influence of previously identified outliers.

The analysis of the Scatterplot Matrix revealed an intriguing finding regarding the classification of objects. Notably, we observed a significant overlap between the quasar class (2) and both the galaxy class (0) and the star class (1) in the plot. However, it is worth mentioning that the star class (1) still exhibited some distinguishable points.

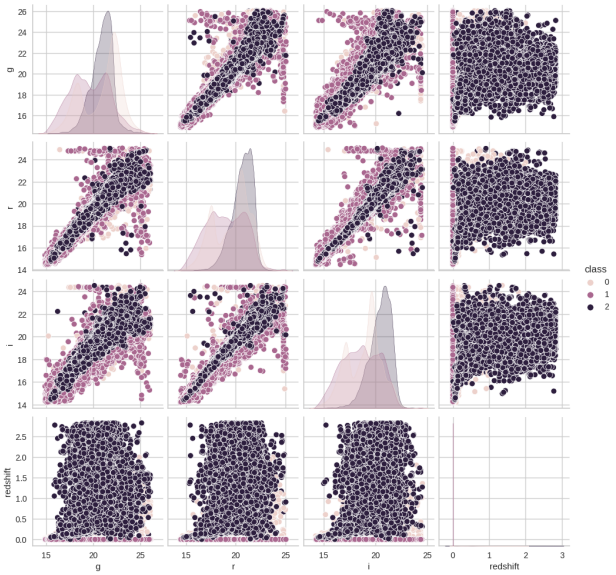


Fig. 6: Scatter plot matrix after removing outliers

To standardize the data, we applied the standard scaler from the scikit-learn library. Standardization is a common data preprocessing technique that transforms the features to have zero mean and unit variance. This normalization process is essential for ensuring fair comparisons and avoiding biases that may arise due to differences in the scales of the features.

By utilizing the standard scaler, we transformed each feature to follow a standard normal distribution, where the mean is 0 and the standard deviation is 1. This normalization technique allows for a more meaningful interpretation of the data and facilitates the training and performance of machine learning models.

To apply the normalization using the standard scaler, we used the following equation:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

where z represents the standard score (or z-score) of a sample, x is the original value of the sample, μ is the mean (average) of the feature, and σ is the standard deviation from the mean.

III. Implemented Machine-Learning Models

A. *Logistic Regression*

Logistic regression is a classification algorithm that predicts the probability that a given instance belongs to one class. In this statistical model, a linear function of the input features is used to model the logit of an instance belonging to a positive class. The logit is then transformed back into a probability using the sigmoid function, which maps any real value into the range [0,1], with the resulting value being interpreted as the predicted probability of belonging to the class.

The goal of the training of the logistic regression model is the optimization of the fit of the possible parameters of the logit function. To determine the best parameters, a cost function, such as the cross-entropy loss function, is minimized, using various optimization algorithms.

For this project, the LogisticRegression class of the sklearn package was used. This class uses the cross-entropy cost function for multi-class classification problems. Three instances of Logistic Regression were tested – without regularization, or with L1 or L2 regularization. Regularization is a method used to prevent overfitting, by reducing the model parameters towards 0, with L1 being Lasso regression, in which some parameter weight coefficients might be reduced to exactly 0 (possibly reducing the number of features tested), while L2 is Ridge Regression, which only reduced the magnitude of the features, without ever fully eliminating them.

Three different optimization algorithms were tested as well:

- SAGA, a stochastic average gradient descent algorithm, uses a running average of past gradient updates to update the weights of the parameters on each iteration, converging faster than other algorithms. [15]
- LBFGS, limited-memory Broyden-Fletcher-Goldfarb-Shanno, a quasi-Newton optimization algorithm that approximates the inverse Hessian matrix of the cost function using a limited memory buffer. [16]
- LIBLINEAR, a library of linear models that uses coordinate descent algorithms, applied only on the L1 and L2 regularization models. This model fits better in situations where the number of features is bigger than the number of examples, and thus isn't expected to perform well on the current dataset. [17]

One of the most important hyper-parameters present on the sklearn package used, and tested on this project, was C, the inverse of regularization strength, that applies the amount of regularization applied to the model, controlling the trade-off between model complexity and the degree to which the model misclassified training data.

B. *SVM*

The sklearn SVC (Support Vector Classifier) is a powerful machine learning algorithm used for classification tasks. It belongs to the family of Support Vector Machines (SVM), which are effective for both linear and non-linear classification. The SVC algorithm works by finding an optimal hyperplane that separates different classes in the feature space, maximizing the margin between the classes.

During training, the SVC identifies support vectors, which are the data points closest to the decision boundaries. These support vectors play a crucial role in defining the hyperplane and making predictions. By considering the support vectors, the SVC can handle complex datasets and achieve good generalization performance. [18]

In this project, we utilized the SVC class from the sklearn package to perform our classification tasks. Several important hyperparameters were considered to configure the SVC model [19]:

- C: The C parameter controls the regularization strength, balancing the desire to minimize training errors with the complexity of the decision function. A smaller C value allows for more margin violations, potentially leading to a simpler decision boundary, while a larger C value enforces a stricter margin, potentially leading to a more complex decision boundary.
- kernel: The kernel parameter specifies the type of kernel function used to transform the input features into higher-dimensional space. Popular choices include linear, polynomial, radial basis function (RBF), and sigmoid kernels. The choice of kernel depends on the nature of the data and the desired decision boundary shape.
- gamma: The gamma parameter influences the reach of each training example and controls the shape of the decision boundary. A small gamma value results in a smoother decision boundary, while a large gamma value can create more complex decision boundaries, potentially leading to overfitting.
- degree: The degree parameter is specific to polynomial kernels and determines the degree of the polynomial function used in the kernel. It controls the flexibility of the decision boundary and should be set based on the complexity of the underlying data.

C. *XG Boost*

XG Boost (Extreme Gradient Boosting) is a scalable, distributed, gradient-boosted decision tree machine learning algorithm. In XG Boost, an ensemble of shallow decision trees is iteratively trained, starting with one single decision tree, with multiple new ones being subsequently added on next iterations. In each iteration, the error residuals of the previous models are used to fit the next model of the tree, with the model trying to focus on the examples in the training set that the previous tree got wrong and trying to correctly classify them in the subsequent trees (boosting). [20]

At each iteration, a loss function is calculated, measuring how well the model performs on the training set. The goal of the algorithm is minimizing this function, through the gradient algorithm, using both L1 and L2 regularization to avoid overfitting. [20]

XG Boost is known for being not only highly accurate, but also scalable, pushing the limits of computing power for boosted tree algorithms, as the XG Boost trees are built in parallel, using multiple CPU cores. XG Boosting also uses techniques such as tree pruning, approximate learning and cache optimization to further enhance the computational power of the model. [20]

Among the most important hyper-parameters that can be defined while building an XG Boost model:

- Gamma, which is the minimum loss reduction required to make a further partition on a leaf node of a tree. It controls the complexity of the decision trees in the model, with higher values reducing the number of splits in the decision tree, and vice-versa. Lower values can lead to overfitting. [21]
- Learning rate, boosting learning rate. It defines how quickly the algorithm learns from the trees added to the model, with lower learning rates meaning taking smaller steps when updating the model, and vice-versa.
- Maximum depth, the maximum tree depth for base learners.
- Number of estimators, that is the number of boosting rounds, or the number of decision trees added to the model during training.

D. *Random Forest*

Random Forest is an ensemble learning algorithm that combines the predictions of multiple decision trees to achieve accurate classifications or predictions.

The primary objective of training a Random Forest model is to create an ensemble of decision trees that collectively provide high predictive accuracy while mitigating overfitting. Each decision tree in the forest is built using a random subset of features and samples from the training dataset.

For the classification tasks in this project, we employed the RandomForestClassifier class from the sklearn package. Several key hyperparameters were carefully considered to configure the Random Forest model. [22] These include:

- n_estimators: This parameter determines the number of decision trees in the forest. Increasing the number of trees can improve the model's performance, but it may also increase computational complexity.
- max_depth: The maximum depth allowed for each decision tree in the forest. By controlling the tree depth, the model can prevent overfitting and generalize well to unseen data.
- min_samples_split: This parameter sets the minimum number of samples required to split an internal node during the tree-building process. Adjusting this parameter can help control the complexity of individual decision trees and prevent overfitting.
- min_samples_leaf: The minimum number of samples required to be at a leaf node. Like min_samples_split, this parameter influences the complexity of the trees and can be adjusted to prevent overfitting.
- max_features: This parameter determines the number of features to consider when looking for the best split at each node. It allows for feature subsampling, which can enhance the diversity among the trees and improve the model's overall performance.

There are a lot of benefits to using Random Forest Algorithm, but one of the main advantages is that it reduces the risk of overfitting and the required training time. Additionally, it offers a high level of accuracy. The Random Forest algorithm runs efficiently in large databases and produces highly accurate predictions by estimating missing data. [23]

E. *Neural Network*

The MLPClassifier, also known as the Multi-Layer Perceptron Classifier, is an artificial neural network algorithm widely used for classification tasks. It employs multiple layers of interconnected neurons to learn complex patterns and relationships within the data. By optimizing the network's weights and biases through the backpropagation algorithm, the MLPClassifier minimizes a cost function during training, enabling it to make accurate predictions. This iterative process adjusts the network's parameters based on the training data, allowing the model to effectively classify new instances. [24]

In this project, the MLPClassifier class from the sklearn package was employed for classification tasks. Several important hyperparameters were considered to configure the MLPClassifier model [25]:

- hidden_layer_sizes: This parameter defines the number of hidden layers and the number of neurons in each hidden layer. By adjusting the architecture of the neural network, we can control its capacity to capture complex patterns in the data.
- activation: The activation function determines the output of a neuron given its inputs. Common choices include the logistic sigmoid function, the hyperbolic tangent function, or the rectified linear unit (ReLU) function. The choice of activation function influences the model's ability to capture nonlinear relationships.
- solver: The solver algorithm determines how the MLPClassifier optimizes its weights and biases. Popular choices include the Stochastic Gradient Descent (SGD) algorithm, Adam optimizer, or Limited-memory BFGS (L-BFGS) algorithm. Each solver has its own advantages and considerations regarding convergence speed and efficiency.
- alpha: This parameter controls the regularization term, which helps prevent overfitting by adding a penalty for large parameter values. It balances the trade-off between model complexity and generalization ability.
- learning_rate: The learning rate determines the step size at each iteration of the optimization process. It affects how quickly the model converges to the optimal solution. Common choices include constant learning rate, adaptive learning rate, or learning rate schedules.
- max_iter: This parameter defines the maximum number of iterations or epochs for training the MLPClassifier. It helps control the training time and prevents the model from training indefinitely.

IV. Model Training Experimental Setup

After finishing the data set analysis, feature engineering and feature normalization, as described in section II, we proceeded to implement the ML models described in section III.

First, we separated the normalized and balanced data into two subsets, according to the holdout method – a training set, and a testing set, with 30% of the examples being attributed into the testing set. The split was made using a fixed random seed, to ensure that the results were reproducible, so that the performance of the different methods employed could be properly compared.

A. *Initial Model Training*

After splitting the data, we took the first step in the model training. To do so, we trained each of the ML models described, using the train dataset. In this initial training, we used the default hyper-parameter values associated with their classes on the sci-kit library (or the XG Boost library, in the case of this model).

Afterwards, we tested the models using the testing dataset, and we evaluated their confusion matrixes and performance metrics, with the results explored on section V.

The parameters used for each of the models are described in Table II.

TABLE II: Initial Parameters

Logistic Regression with No Regularization	
Maximum iterations	1000
C	1.0
Solving algorithm	SAGA
Logistic Regression with L1 Regularization	
Maximum iterations	5000
C	1.0
Solving algorithm	SAGA
Logistic Regression with L2 Regularization	
Maximum iterations	1000
C	1.0
Solving algorithm	SAGA
SVM	
C	1.0
Kernel	Radial basis function (RBF)
Degree	3
Gamma	$\frac{1}{n_{\text{features}} \cdot X.\text{var}()}$
XG Boost	
Gamma	0
Learning Rate	0.1
Maximum depth	3
Number of estimators	1000
Random Forest	
Maximum depth	None
Max features	Square Root
Minimum samples split	2
Minimum samples leaf	1
Number of estimators	100
Neural Network	
Activation function for hidden layer	Rectified linear unit function (relu)
Alpha	0.0001
Hidden Layer Sizes	(4,4)
Learning Rate	Constant
Maximum Iterations	750
Solving Algorithm	ADAM

For the logistic regression with L1 regularization, we had initially used 1000 as the value for the maximum iterations hyper-parameter, as was the case with the other types of logistic regression. However, we switched to 5000 as the model never converged when using 1000 as the value for this parameter. For the same reasons, we switched from the default 200 to 650 maximum iterations when training the neural network.

B. *Hyper-Parameter Tuning*

For the next step of the models training, we did hyper-parameter tuning. To do so, for each of the developed models we performed grid search K-fold cross-validation, using sklearn's GridSearchCV class.

Grid search is a technique used to find the best set of hyper-parameters for an ML model, by exhaustively searching over a predefined hyper-parameter space. [26] Meanwhile, K-fold cross validation is a technique that allows the evaluation of a ML model's performance, by splitting the data into K-equally sized subsets, and training and evaluating the model in each one of these subsets. [27] The combination of both techniques allows the use of K-fold cross validation to evaluate each of the tested hyper-parameters in grid search. In this project, 5 subsets were used for cross validation.

However, this task can be computationally expensive and take some time to run, particularly when the hyper-parameter space explored is vast. Thus, for each of the ML models, we focused only on the exploration of the most important hyper-parameters, using rather small subsets of values that encompass not only the default parameters already used before, but also the parameters found on other papers and studies we found while searching for related work to this project.

The summary of the hyper-parameter intervals tested for each model is found on Table III.

At the end of this step, we saved the optimal hyper-parameters for each of the ML models, with these results being analyzed on Section V.

For the random forest, we decided not to use the default Non parameter for the maximum depth of the tress, as we thought this could make the training take a much bigger execution time (as it was, the grid search on Random Forest took over 10 hours of execution time).

TABLE III: Hyper-Parameter Tuning Parameters

Logistic Regression with No Regularization	
Maximum iterations	200, 500, 1000, 1500
C	1, 10, 100, 1000, 10000
Solving algorithm	SAGA, LBFGS and LIBLINEAR
Logistic Regression with L1 Regularization	
Maximum iterations	200, 500, 1000, 1500
C	1, 10, 100, 1000, 10000
Solving algorithm	SAGA and LIBLINEAR
Logistic Regression with L2 Regularization	
Maximum iterations	200, 500, 1000, 1500
C	1, 10, 100, 1000, 10000
Solving algorithm	SAGA, LBFGS and LIBLINEAR
SVM	
C	1, 10, 100, 1000
Kernel	RBF and sigmoid
Gamma	1, 10, 100, 1000
XG Boost	
Gamma	0, 0.1, 0.3
Learning Rate	0.01, 0.1, 0.3
Maximum depth	3, 4, 5
Number of estimators	100, 1000
Random Forest	
Maximum depth	5, 10, 12, 15
Max features	Square Root, Logarithmic
Minimum samples split	2, 10, 14
Minimum samples leaf	2, 4, 6
Number of estimators	100, 1000
Neural Network	
Activation function for hidden layer	relu and tanh
Alpha	0.0001, 0.001, 0.01
Hidden Layer Sizes	(4,4); (8,); (8,6)
Learning Rate	Constant and adaptive
Solving Algorithm	ADAM and SGD

C. *Optimal Model Training*

The third and final step was the retraining of each of the models using their optimized hyper-parameters and using 5-fold cross validation to evaluate the performance of each one.

To do so, we saved the best estimator obtained in the 5-fold cross validation. Then, we tested that estimator using the test set. After that, we analyzed the confusion matrix, the main performance metrics, as well as the ROC/AUC curve, the precision recall curve, and the class prediction error graphic for the best estimators of each model. These metrics are further explained in the next section. Meanwhile, the results obtained are explored in Section V of the report.

D. *Performance Evaluation Metrics*

To compare the performance of the various optimized ML models, while testing with the test set, the models were evaluated based on some performance metrics. Most of these metrics were based on the obtained confusion matrixes, and are listed below:

- Accuracy, the number of correctly classified instances divided by the total number of instances.
- Recall, the true positive rate.
- Precision, the fraction of correctly classified positive samples from all the samples classified as positive.
- F1-score, which determines the weighted average of Precision and Recall
- Balanced accuracy, which calculates the average of the per-class accuracy, giving equal weight to each class, no matter whether the set is balanced or not.

Even though the initial dataset was balanced using SMOTE oversampling, as previously described, we still felt like balanced accuracy was a better basis for comparing the different models, instead of using accuracy, and thus this was selected as the main performance metric for model comparisons. Besides these metrics, the final optimized models were also evaluated using the following techniques:

- ROC curve, a curve obtained by plotting the true positive rate against the false positive rate for a classifier at a variety of thresholds. Related to this curve is the AUC, area under curve, which can be used as a score to compare different classifiers – the closer to 1.0 the better the predictive power.
- Precision-Recall curve, which summarizes the trade-off between precision and the recall for different classification thresholds. Again, models with better predictive power should have high areas under the curve, symbolizing a system with high recall and precision, returning many correctly labeled results.
- Class Prediction Error, a graph that allows the visualiza-

tion of the number of examples mislabeled for each of the classes tested on the data set.

V. Results and Analysis

In the following section, we will discuss the results obtained during the model training process. We will first present the results of the initial training, followed by the results of the hyper parameter tuning, and of the final training of the optimized models.

We will then compare the results of the different models with each other, as well as with previous results found on the related work. To do so, we will use the performance metrics described on the previous section.

A. *Initial Training Results*

As previously described, the first step in the training of the ML models was to train them using the default hyper-parameters previously presented on Section IV. To evaluate each of these models, we tested them with the training set, and we obtained the resulting confusing matrixes, presented on Figure 7, as well as the performance metrics, presented on Table V.

As we can see analyzing the obtained performance metric results in Table V, the values for the performance metrics are all above 90%, with XG Boost and Random Forest reaching a balanced accuracy value of 97%. On the other hand, SVM was the weakest model, with 91% balanced accuracy, and being the only value where some of the metrics scored lower than 90%. As theorized in Section I, XG Boost appears to be a good improvement when compared to the other models already trained on the related work.

It's also noticeable that the Precision, Recall and F1-Scores are all higher for the classification of Stars, when compared with the classification of Galaxy or Quasar examples. In most models, these values reach 100%. After analyzing these results, we proceeded to do hyper-parameter tuning.

(a) Logistic Regression with No Regularization

(b) Logistic Regression with L1 Regularization

(c) Logistic Regression with L1 Regularization

(d) SVM

(e) XG Boost

(f) Random Forest

(g) Neural Network

Fig. 7: Confusion Matrix for the Initial Training

B. *Hyper Parameter Tuning*

Hyper parameter tuning was performed with grid search cross-validation, according to the plan specified on Section IV. The goal was to obtain the optimal hyper-parameters for the performance of each trained ML model.

In order to evaluate which combination of hyper-parameters was best, the models developed during the grid search CV were evaluated with the performance metrics previously described. However, since we then retrained the optimal models on the following step of the experimental procedure, in this section we will only present the optimal hyper-parameters obtained for each model.

When analyzing the optimal parameters obtained, present in Table IV, we can notice some major changes in some of the models. For example, all logistic regression models obtained optimal values for the C parameter much higher than the default 1.0. Also, LBFGS was favored as the solving algorithm over SAGA, except for the L1 regularization model.

Other important changes compared with the parameters used on the initial training include the C value for the SVM model, the number of estimators and the use of the logarithmic function for the determination of maximum features for the Random Forest, and the increased number of neurons on the hidden layer size, as well has the learning rate mode, for the neural network.

C. *Optimal Model Training Results*

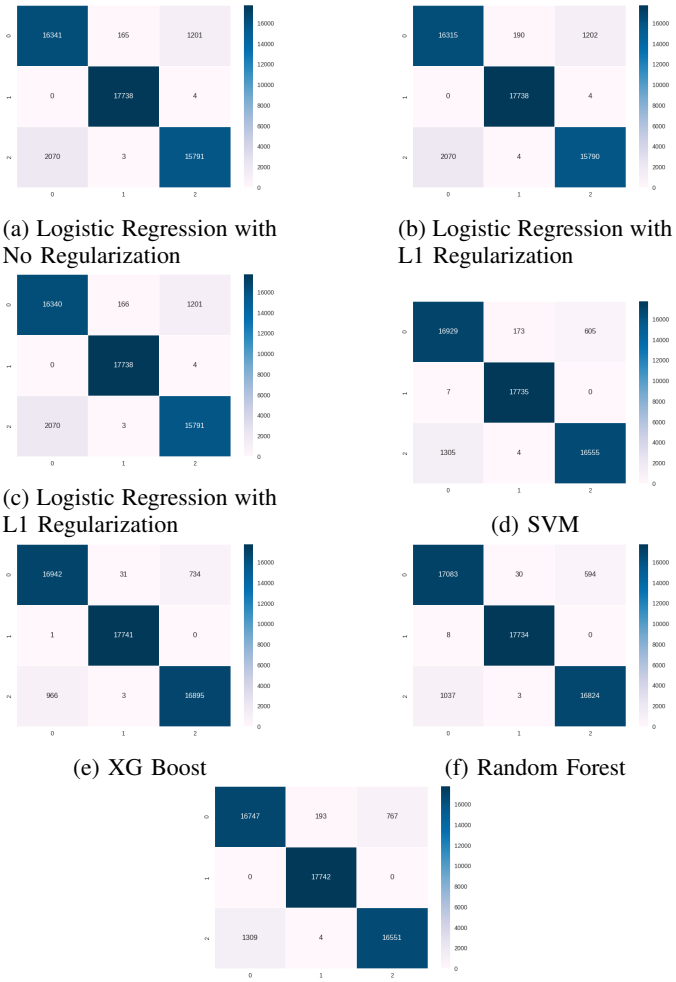
As explained on Section IV, we then retrained each ML model with their optimal hyper-parameters, using 5-fold cross validation, to obtain the final models for this project. We then evaluated the best estimator for each model using, once again, the confusion matrix and the performance metrics, as well as the ROC curve, Precision-Recall Curve, and the Class Prediction Error graph.

It's worth pointing out, that it wasn't possible obtain the Precision-Recall Curve and the Class Prediction Error graph for the XG Boost, due to technical errors.

With a first glance at both the confusion matrixes and the performance metrics of the final optimized models, available

TABLE IV: Hyper-Parameter Tuning Optimal Parameters

Logistic Regression with No Regularization	
Maximum iterations	200
C	10000
Solving algorithm	LBFGS
Logistic Regression with L1 Regularization	
Maximum iterations	5000
C	100
Solving algorithm	SAGA
Logistic Regression with L2 Regularization	
Maximum iterations	500
C	5000
Solving algorithm	LBFGS
SVM	
C	1000
Kernel	RBF
Gamma	1
XG Boost	
Gamma	0
Learning Rate	0.3
Maximum depth	5
Number of estimators	1000
Random Forest	
Maximum depth	15
Max features	Logarithmic
Minimum samples split	2
Minimum samples leaf	2
Number of estimators	1000
Neural Network	
Activation function for hidden layer	tanh
Alpha	0.0001
Hidden Layer Sizes	(8,6)
Learning Rate	adaptive
Solving Algorithm	ADAM



(a) Logistic Regression with No Regularization

(b) Logistic Regression with L1 Regularization

(c) Logistic Regression with L1 Regularization

(d) SVM

(e) XG Boost

(f) Random Forest

(g) Neural Network

Fig. 8: Confusion Matrix for the optimal models

on Figure 7 and Table VI, it's possible to check that for the majority of the models the metrics have more or less the same values, with only a few exceptions worthy of notice.

In the case of the Logistic Regression models, the balanced accuracy stayed the same value (94%), with the exception being for the L1 regularization model, where the accuracy decreased from 95% to %. Precision and Recall scores are lower for the classification of galaxies and quasars in all 3 models (with changes from the 92%-94% mark to 88%-90%), while the scores for stars went up (reaching 100% in nearly all metrics), with the F1-Score increasing for all 3 classes.

For the XG Boost and the Random Forest models, the balanced accuracy was 97% in both cases, staying the same as with the pre-optimization training. The precision, recall and F1-Score values for galaxies and quasars slightly lowered, as was the case with Logistic Regression, from around the 96%-97% mark, to 94%-95%.

The Neural Network and SVM however, bucked these trends. For the Neural Network, the balanced accuracy increased from 95% to 96%. The values of Precision, Recall and F1-Score stayed mostly the same, as opposed to what happened in the previous models.

Meanwhile, for SVM, there was a more considerable increase in balanced accuracy, from 91% to 96%, with hyper-parameter optimization making this model more able to compete with XG Boost and Random Forest. There was also considerable increase on the Precision, Recall and F1-Scores, even for the galaxies and quasar examples.

When looking at the ROC curves, present in Figure 9 it's very visible that all models have high ture positive rates (TPR) and low false positive rates (FPR), has all models quickly reach the 100% AUC mark. However, XG Boost and Random Forest clearly reaches that point quicker, at a relatively low threshold value, whereas Logistic Regression and SVM need slightly higher values to reach that point. The analysis of the Precision-Recall curves, present in Figure 10 yields similar conclusions, with the Logistic Regression having slightly worse models. However, all of the models have good looking curves, meaning they all have high recall, identifying positive examples, while maintaining high precision, that is, identifying a low number of false positives.

By analyzing the Class Prediction Error graph, on Figure 11 we can confirm that indeed all models classify the Star examples nearly 100% correctly, with the mislabeling of examples occurring with the mixing of Galaxy and Quasar examples. It's also quite visible that the Logistic Regression models had more instances of misclassification.

TABLE V: Performance Metrics for Initial Training

Model	Precision			Recall			F1-Score			Bal. Accuracy
	Galaxy	Star	QSO	Galaxy	Star	QSO	Galaxy	Star	QSO	
Log Reg No Rg	0.92	0.97	0.95	0.92	1.00	0.92	0.92	0.98	0.93	0.94
Log Reg L1 Rg	0.93	0.98	0.95	0.94	1.00	0.92	0.93	0.99	0.94	0.95
Log Reg L2 Rg	0.92	0.97	0.95	0.92	1.00	0.92	0.92	0.98	0.93	0.94
SVM	0.92	0.87	0.97	0.83	1.00	0.92	0.88	0.93	0.95	0.91
XG Boost	0.96	1.00	0.97	0.97	1.00	0.96	0.97	1.00	0.96	0.97
Random Forest	0.96	1.00	0.97	0.97	1.00	0.96	0.97	1.00	0.97	0.97
Neural Network	0.93	0.99	0.95	0.94	1.00	0.92	0.93	0.99	0.93	0.95

TABLE VI: Performance Metrics for the Optimal Models Training

Model	Precision			Recall			F1-Score			Bal. Accuracy
	Galaxy	Star	QSO	Galaxy	Star	QSO	Galaxy	Star	QSO	
Log Reg No Rg	0.89	0.99	0.93	0.92	1.00	0.88	0.90	1.00	0.91	0.94
Log Reg L1 Rg	0.89	0.99	0.93	0.92	1.00	0.88	0.90	0.99	0.91	0.94
Log Reg L2 Rg	0.89	0.99	0.93	0.92	1.00	0.88	0.90	1.00	0.91	0.94
SVM	0.93	0.99	0.96	0.96	1.00	0.93	0.94	0.99	0.95	0.96
XG Boost	0.95	1.00	0.96	0.96	1.00	0.95	0.95	1.00	0.95	0.97
Random Forest	0.94	1.00	0.97	0.96	1.00	0.94	0.95	1.00	0.95	0.97
Neural Network	0.93	0.99	0.96	0.95	1.00	0.93	0.94	0.99	0.94	0.96

*D. **Analysis of Results***

Looking at the set of results obtained for all models, both the results for the initial training with default parameters, as well as the results for the optimal model training after hyper-parameter tuning, it's quite noticeable that all the models have satisfying results when it comes to their performance metrics, and to the accuracy with which they classify the different examples in the testing set.

All of the models either stayed the same, or slightly improved on most performance metrics, in particular the balanced accuracy, when comparing the pre-hyper-parameter tuning training present in Table V, and the post hyper-parameter tuning training in Table VI, with the exception being the 3 Logistical Regression models, that performed relatively worse. SVM, on the other hand, was the model whose accuracy and performance increased more after tuning, going from the worst model, to being able to compete with XG Boost and the Random Forest models for accuracy.

Even though all models got better performance metrics when classifying Stars, reaching the 100% value in nearly all Precision, Recall and F1-Score metrics for this class of examples, the performance of nearly all models, except for SVM, slightly decreased with hyper-parameter tuning, when classifying Galaxy and Quasar examples. It's worth mentioning at this point that the testing set was confirmed to also be well balanced, even after the application of the holdout method, as the output for the performance classification revealed that there were equal instances of examples for the 3 classes.

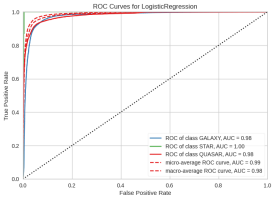
There are a few possibilities that might explain the reason why the Logistic Regression models performance decreased after hyper-parameter tuning, whereas the performance of models such as XG Boost or Random Forest didn't increase quite as expected. One of them, is the possibility of overfitting, as the same validation set was used for hyper-parameter tuning and model evaluation. This is a strong possibility in the case of the Logistic Regression models, as maybe further feature engineering through polymerization could have been tested to see if their results improved.

Another possibility is that the set of values tested for each hyper-parameter of each ML model wasn't varied enough. We didn't test has many combinations of hyper-parameters as we wanted, because the execution of grid search cross-validations is quite a time-intensive and computationally heavy laborious effort. This is true in particular for more complex models, like Random Forest or XG Boost, that have a lot of complex hyper-parameters that could have been further optimized. However, we are still happy with the values obtained after the hyper-parameter tuning, as all models had very high accuracy values for all performance metrics.

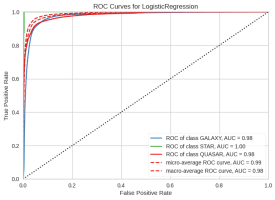
When comparing the different model's performance, it's quite obvious that XG Boost and Random Forest offer the highest performance, both before and after the hyper-parameter tuning. This is as expected, as Random Forest was the model with better results the previous works done on this dataset, and explored in Section I. Meanwhile, the effort to include XG Boost, that wasn't present in these related works, proved a very worthy endeavour, as this model reached the highest overall performance metrics, classifying nearly 100% of the examples on training set correctly. On the other hand, the Logistic Regression models performed noticeably worse than the rest, with no visible difference being made between the use of L1, L2 or no regularization. SVM was clearly the model benefitted the most from hyper-parameter tuning, increasing successfully all the performance metrics.

However, as previously mentioned, all models have very satisfying performance metrics, ROC curves and Precision-Recall curves, and thus all of them could be applied on machine learning problems regarding stellar classification.

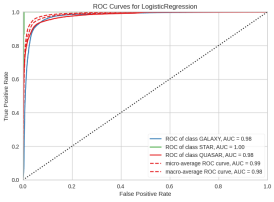
It's also clearly noticeable that the main confusion for all



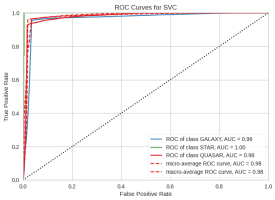
(a) Logistic Regression with No Regularization



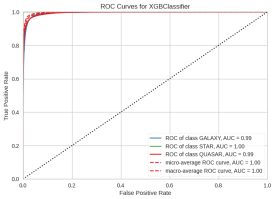
(b) Logistic Regression with L1 Regularization



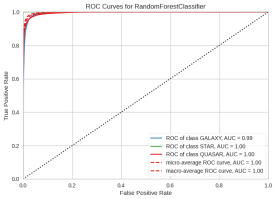
(c) Logistic Regression with L1 Regularization



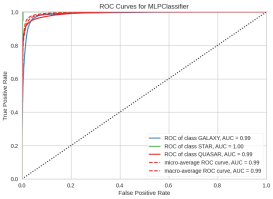
(d) SVM



(e) XG Boost

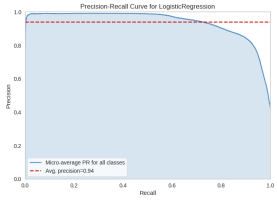


(f) Random Forest

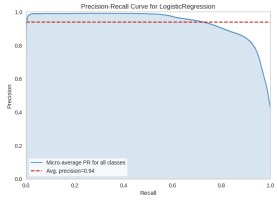


(g) Neural Network

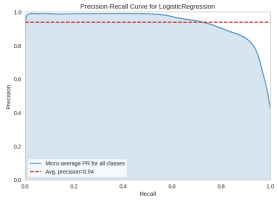
Fig. 9: ROC Curves for the final Optimal Models



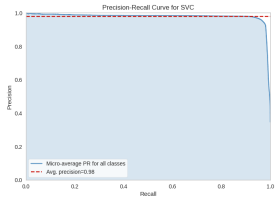
(a) Logistic Regression with No Regularization



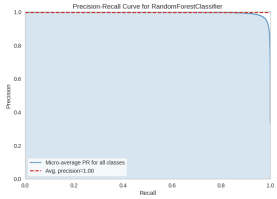
(b) Logistic Regression with L1 Regularization



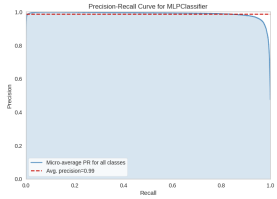
(c) Logistic Regression with L1 Regularization



(d) SVM



(e) Random Forest



(f) Neural Network

Fig. 10: Precision-Recall Curves for the final Optimal Models

models is between the classification of Galaxies and Quasars, with most of the misclassification of examples happening in these classes. Given that quasars are super-galaxies in an early development stage, this might mean that the photometric features of both are quite similar, having overlap in their values. Thus, further studies on this field should try to include more examples of these classes, and the differences of the redshift and the photometric features of both should be further studied to accomplish better results, and maybe discriminate potential new features for the distinction between both.

The comparison between the results obtained in this project, and thre results obtained for the projects mentioned in related work, is present on Tables VII for the SVM models, and Table VIII for the Random Forest models. As previously stated, none of the projects had worked with XG Boost or Neural Network, while only one project, by Omat et Al. [7], having used a multinomial version of Logistic Regression, with disappointing values when compared with other models, as was the case in our project. In these projects, the main performance metric used was accuracy instead of balanced accuracy; otherwise, they all published results for the Precision, Recall and F1-Score metrics. In these papers, the Random Forest model also obtained better results, when compared with the SVM model, like in the present project. All projects obtained around 100% value for the Precision, Recall and F1-Scores metrics when classifying Stars. However, all of them had slightly lower performance for the classification of Galaxies and Quasars, corroborating the results found on this project. The values found for these metrics were 1%-2% bigger than the results obtained in the current project, maybe due to the overfitting or lack of variety in the set of values tested for hyper-parameter tuning, previously described.

VI. Conclusions

In this project, the SDSS17 Stellar Classification dataset was used to train five different machine learning (ML) models: Logistic Regression (with or without regularization), SVM, Neural Network, XG Boost, and Random Forest. The objective was to classify samples as Stars, Galaxies, or Quasars based on their spectral characteristics obtained from photometric system features and redshift.

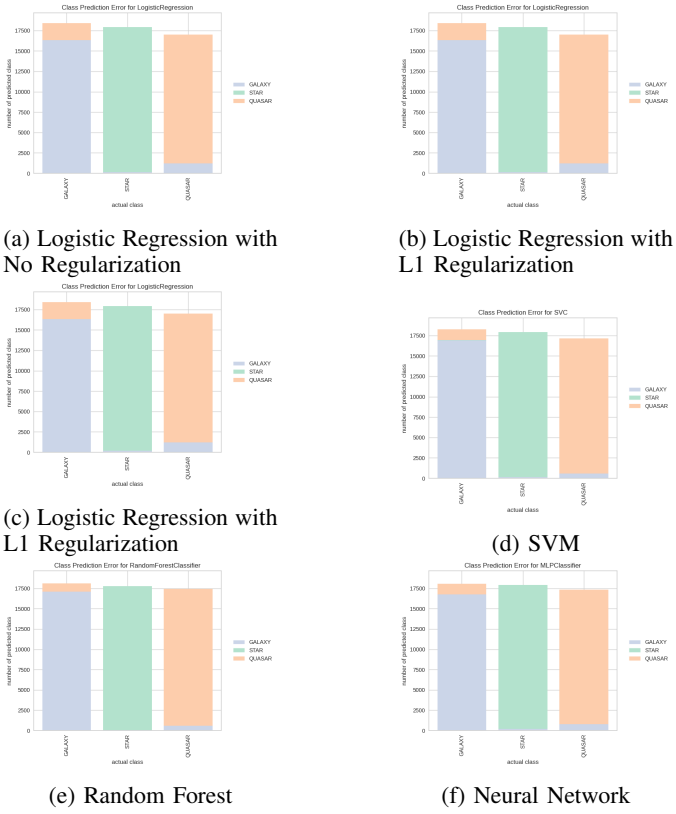
All the models achieved accurate classifications and demonstrated high performance metrics. However, the most accurate models were XG Boost and Random Forest, both achieving a balanced accuracy of 97% and Precision, Recall, and F1-Score values ranging from 95% to 100%. Logistic Regression performed the worst with a balanced accuracy value of 94%

TABLE VII: Comparison with SVM Models from Related Work

Model	Precision			Recall			F1-Score			Bal. Accuracy	Accuracy
	Galaxy	Star	QSO	Galaxy	Star	QSO	Galaxy	Star	QSO		
This Project	0.93	0.99	0.96	0.96	1.00	0.93	0.94	0.99	0.95	0.96	–
Omat et Al. [7]	0.97	1.00	0.96	0.99	1.00	0.92	0.98	1.00	0.94	–	0.98
Z. Qi. [8]	0.95	0.97	0.98	0.95	1.00	0.97	0.97	1.00	0.98	–	0.98
Kaggle Notebook [10]	0.95	0.97	0.98	0.96	1.00	0.96	0.96	0.99	0.97	–	0.96

TABLE VIII: Comparison with Random Forest Models from Related Work

Model	Precision			Recall			F1-Score			Bal. Accuracy	Accuracy
	Galaxy	Star	QSO	Galaxy	Star	QSO	Galaxy	Star	QSO		
This Project	0.94	1.00	0.97	0.96	1.00	0.94	0.95	1.00	0.95	0.97	–
Omat et Al. [7]	0.97	1.00	0.96	0.99	1.00	0.92	0.98	1.00	0.94	–	0.98
Z. Qi. [8]	0.97	1.00	0.98	0.98	1.00	0.97	0.97	1.00	0.98	–	0.98
Kaggle Notebook [10]	0.97	1.00	0.98	0.98	1.00	0.97	0.98	1.00	0.98	–	0.98



(a) Logistic Regression with No Regularization

(b) Logistic Regression with L1 Regularization

(c) Logistic Regression with L1 Regularization

(d) SVM

(e) Random Forest

(f) Neural Network

Fig. 11: Class Prediction Graphs for the final Optimal Models

(with or without regularization) and Precision, Recall, and F1-Score values in the range of 89% to 93%.

These findings align with related work, where Random Forest consistently outperformed other models. XG Boost, on the other hand, proved to be a valuable addition to the project as it had not been tested on this dataset before.

The models achieved excellent classification results for Stars, with performance metrics of 99% or 100%. However, most misclassifications occurred between Galaxies and Quasars. This observation supports previous research, indicating that the evaluated features, particularly the photometric parameters, do not differentiate well between Quasars and Galaxies. To improve classification accuracy, it is recommended to collect and measure more samples of Galaxy and Quasar observations while exploring additional features.

Hyperparameter tuning significantly improved the accuracy of SVM, allowing it to compete with Random Forest and XG Boost. However, it only had a slight impact on the performance of other models and even decreased the metrics for Logistic Regression. This might be attributed to overfitting or insufficient exploration of hyperparameter values during the grid search cross-validation process. Future work should involve testing a broader range of hyperparameters for optimization.

In conclusion, XG Boost and Random Forest models are recommended for stellar classification problems. However, XG Boost is preferred due to its computational and time efficiency.

REFERENCES

[1] "Stellar Classification Dataset - SDSS17 — Kaggle." *Kaggle*, [Online]. Available: https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss17 (accessed May 08, 2023).

[2] "Sloan Digital Sky Survey." *SDSS*, [Online]. Available: https://classic.sdss.org/ (accessed May 08, 2023).

[3] "Completed Spectroscopic Surveys – SDSS." *SDSS*, [Online]. Available: https://www.sdss.org/dr18/completed_surveys/ (accessed May 08, 2023).

[4] "Data Release 17 — SDSS." *SDSS*, [Online]. Available: https://www.sdss4.org/dr17/ (accessed May 08, 2023).

[5] "ESA - What is 'red shift'?" *ESA*, [Online]. Available: https://www.esa.int/Science_Exploration/Space_Science/What_is_red_shift (accessed May 08, 2023).

[6] "EarthSky — What is a quasar?" *EarthSky*, [Online]. Available: https://earthsky.org/astronomy-essentials/definition-what-is-a-quasar/ (accessed May 08, 2023).

[7] D. Omat, J. Otey, and A. Al-Mousa, "Stellar Objects Classification Using Supervised Machine Learning Techniques," in *Proceedings - 2022 23rd International Arab Conference on Information Technology, ACIT 2022*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 1-5. doi: 10.1109/ACIT57182.2022.9994215.

[8] Z. Qi, "Stellar Classification by Machine Learning," *SHS Web of Conferences*, vol. 144, p. 03006, 2022. doi: 10.1051/shsconf/202214403006.

[9] "Stars & Galaxies: EDA and Classification — Kaggle." *Kaggle*, [Online]. Available: https://www.kaggle.com/code/psycon/stars-galaxies-eda-and-classification (accessed May 08, 2023).

[10] "Stellar Classification - 98.4% Acc 100% AUC — Kaggle." *Kaggle*, [Online]. Available: https://www.kaggle.com/code/beyzanks/stellar-classification-98-4-acc-100-auc (accessed May 08, 2023).

[11] Abdurro'uf et al., "The Seventeenth Data Release of the Sloan Digital Sky Surveys: Complete Release of MaNGA, MaStar, and APOGEE-2 Data," *Astrophys J Suppl Ser*, vol. 259, no. 2, p. 35, Apr. 2022. doi: 10.3847/1538-4365/ac4414.

[12] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal Of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2011. doi: 10.1613/jair.953.

[13] "Feature Selection Techniques. Feature Selection Techniques — by Sangita Yemulwar — Analytics Vidhya — Medium," *Medium*, [Online]. Available: https://medium.com/analytics-vidhya/feature-selection-techniques-2614b3b7efcd (accessed May 08, 2023).

[14] "InterQuartile Range (IQR)," *Boston University School of Public Health*, [Online]. Available: https://sphweb.bumc.bu.edu/otlt/mph-modules/bs/bs704_summarizingdata/bs704_summarizingdata7.html (accessed May 08, 2023).

[15] A. Defazio, F. Bach, and S. Lacoste-Julien, "SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives," in *Adv Neural Inf Process Syst*, vol. 2, no. January, 2014, pp. 1646–1654. [Online]. Available: https://arxiv.org/abs/1407.0202v3. (accessed May 08, 2023).

[16] "Limited-memory BFGS - Wikipedia," *Wikipedia*, [Online]. Available: https://en.wikipedia.org/wiki/Limited-memory_BFGS (accessed May 08, 2023).

[17] "LIBLINEAR – A Library for Large Linear Classification," *National Taiwan University*, [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/liblinear/ (accessed May 08, 2023).

[18] "Diving into C-Support Vector Classification — by Gustavo Santos — Towards Data Science," *Towards Data Science*, [Online]. Available: https://towardsdatascience.com/diving-into-c-support-vector-classification-221ced32e4b4 (accessed May 08, 2023).

[19] "sklearn.svm.SVC — scikit-learn 1.2.2 documentation," *scikit-learn*, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (accessed May 08, 2023).

[20] "XGBoost – What Is It and Why Does It Matter?" *NVIDIA*, [Online]. Available: https://www.nvidia.com/en-us/glossary/data-science/xgboost/ (accessed May 08, 2023).

[21] "A Gentle Introduction to XGBoost for Applied Machine Learning - MachineLearningMastery.com," *Machine Learning Mastery*, [Online]. Available: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/ (accessed May 08, 2023).

[22] "sklearn.ensemble.RandomForestClassifier — scikit-learn 1.2.2 documentation," *scikit-learn*, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (accessed May 08, 2023).

[23] "Random Forest Algorithm," *Simplilearn*, [Online]. Available: https://www.simplilearn.com/tutorials/machine-learning-tutorial/random-forest-algorithm (accessed May 08, 2023).

[24] "Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis — by Carolina Bento — Towards Data Science," *Towards Data Science*, [Online]. Available: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141 (accessed May 08, 2023).

[25] "sklearn.neural_network.MLPClassifier — scikit-learn 1.2.2 documentation," *scikit-learn*, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html (accessed May 08, 2023).

[26] "What Is Grid Search?. Explaining How To Obtain Optimal… — by Farhad Malik — FinTechExplained — Medium," *Medium*, [Online]. Available: https://medium.com/fintechexplained/what-is-grid-search-c01fe886ef0a (accessed May 08, 2023).

[27] D. R. Roberts et al., "Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure," *Ecography*, vol. 40, no. 8, pp. 913–929, Aug. 2017. doi: 10.1111/ECOG.02881.