Investigating the Efficacy of Genetic Algorithms for Hyperparameter Optimization in Star-Galaxy Classification

M.Sc Data Science Dissertation

16 August 2023

Pratik Bhushan Barve
Student Number: C00290846

Supervisor: Paul Barry

Master of Science in Data Science
Course Code: KCDAR_M_Y5
Department of Computing
South East Technological University, Carlow
Ireland

Word Count: 11123

Dedicated to my parents

Anuradha & Bhushan

DECLARATION

* I declare that all of the content in this work, with the exception of those duly acknowledged, is entirely mine.
* I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams, or other material; including software and other electronic media in which intellectual property rights may reside.
* I have provided a complete bibliography of all works and sources used in the preparation of this submission.
* I am aware that failing to adhere to the Institute's anti-plagiarism policies is a serious offense.

Student Name: Pratik Bhushan Barve

Student Number: C00290846

Signature:

Date: 16^{th} August 2023

ACKNOWLEDGEMENTS

First and foremost I would like to thank professor Paul Barry for being an amazing supervisor and one of the best teachers I have known. Your guidance and kind words kept me motivated throughout the course and especially during thesis.

I would like to extend my heartfelt gratitude to Guido Van Rossum for his pioneering contributions to the development of Python programming language. He inspired a generation of people willing to contribute to open-source projects which enables us to use python and it's libraries free of cost.

I would like to thank the SDSS consortium for providing the dataset free of cost to the public.

I would like to thank the MSc Data Science class of 2023 for being an amazing group of friends.

I would like to thank Professor Madhuri Sawant, my friends Jigar, Pradyumnya for their support/guidance/feedback during the thesis.

I would also like to thank my cousin Siddhi for her support and feedback.

Finally, I am forever indebted to my parents who supported me emotionally, financially, in every way and I thank you for everything. All of your sacrifices, hardships and efforts are acknowledged. I am what I am because of you.

And last but not the least, I would like to thank my partner Vaidehi for being my pillar and supporting me in every way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the ever-expanding field of artificial intelligence, machine learning has achieved remarkable breakthroughs across a wide spectrum of applications. In recent years, the landscape of machine learning has undergone remarkable transformation, ushering in a new era of sophisticated models capable of tackling intricate challenges across diverse domains. These advancements have emerged as a direct consequence of the field's progress, allowing for the development of complex solutions that were previously unattainable. Amid this evolution, the classification of astronomical objects has emerged as a domain of paramount significance. The precise identification of stars, galaxies, and quasars assumes a pivotal role in advancing our comprehension of the universe's inner workings.

1.1 Motive

In the Research Methods module, the class was given an assignment of writing a research proposal as part of our learning objectives. Students were provided with a list of potential topics to help them find something they are interested in or might find interesting. Among the topics was Genetic Algorithms, which caught my attention since it was something I had not heard before. In order to learn more about this topic in depth, I decided to prepare a research proposal around the topic in order to get the opportunity to immerse myself in it more deeply. Based on my literature review, I found that genetic algorithms are extensively used to solve optimization problems. My interest was piqued by a paper which used genetic algorithms in order to fine-tune hyperparameters for machine learning models. Due to my background in astronomy, I was intrigued by the idea that genetic algorithms could be used to tune hyperparameters of models that are used in astronomy. The study would be extremely exciting because of its highly interdisciplinary nature. I find it fascinating when concepts and ideas give inspiration to concepts across different disciplines.

1.2 Sloan Digital Sky Survey

The Sloan Digital Sky Survey (SDSS) is an all-sky imaging survey that uses a large telescope measuring 2.5 meters in diameter to take pictures of the night sky. One of the best aspects of SDSS is the free public accessibility of the data. Very few research papers were published in the coinciding fields

of computer science and astronomy which utilised the recently released new dataset - DR18 (Data Release 18). Wierzbiński et al. (2021) did a very similar study as this thesis but with a small dataset and the older version of data. I decided to build up on it and use a much larger dataset from the latest release and try to improve upon model performance by tuning the hyperparameters and/or validate the results of this paper.

Figure 1.1: The SDSS 2.5 meter telescope (*Sloan Digital Sky Survey | Alfred P. Sloan Foundation*, n.d.)

1.3 Aim and objective

Traditionally, the process of fine-tuning machine learning model hyperparameters has been a manual endeavor, involving human intervention to optimize performance. While this method has yielded notable outcomes, it bears the drawbacks of being labor-intensive, time-consuming, and susceptible to subjective biases. Genetic algorithms, underpinned by the evolutionary principles of natural selection, emerge as a class of optimization techniques that mimic genetic recombination and mutation. These algorithms iteratively refine and adapt solutions across multiple generations by encapsulating hyperparameters of machine learning models within a population of prospective solutions.

The research goal of this thesis is to use genetic algorithms to tune machine learning hyperparameters and to quantitatively evaluate the performance of machine learning models. Does genetic optimisation work effectively to tune hyperparameters and improve model performance? Does implementation of genetic algorithm incurr high computational costs? This will help us decide if it is feasible to use this algorithm or not. These are the research questions that I aim to answer.

1.4 Structure of this document

I have structured the document in a way which will help the reader comprehensively explore the research undertaken. The next chapter is indepth literature review which summarises the current knowledge about all the concepts, tools and technolgies relevant to the thesis. It commences by delving into the SDSS's role in the realm of astronomical observation, followed by a comprehensive survey of previous works in this domain. The chapter further delves into machine learning, offering detailed insights into various techniques such as Random Forest Classifier, Gradient Boosting Classifier, Logistic Regression, Support Vector Machines, Cross Validation, Overfitting, Underfitting, and Machine Learning Metrics.

This is followed by chapter titled Methodology which gives a roadmap of research method that was implemented. It initiates with the intricacies of data collection, followed by a thorough examination of preprocessing and feature selection techniques. The establishment of baseline performance serves as a pivotal point before transitioning into the core of the methodology—the implementation of Genetic Algorithms. The chapter concludes by outlining the evaluation process and criteria used to gauge the results.

Observations and findings chapter begins with an in-depth exploration of Exploratory Data Analysis (EDA), uncovering insights from the data. It then systematically dissects the performance of each model employed, the Random Forest Classifier, Gradient Boost Classifier, and Logistic Regression.

Finally, Chapter 5 draws conclusions, it summarizes the findings, creating a coherent narrative that addresses the research objectives. The conclusions offer a succinct summary of the research's contributions and implications, while also acknowledging areas of potential expansion and further exploration.

Chapter 2

Literature Review

2.1 Sloan Digital Sky Survey (SDSS)

The Sloan Digital Sky Survey (SDSS) is a large scale multi spectral imaging and spectroscopic redshift survey using a dedicated 2.5 meter wide-angle optical telescope at Apache Point Observatory in New Mexico, USA. It began its operation in the year 2000 and since then have been taking continuous wide angle high resolution images of the sky. The telescope uses a camera consisting of 30 charge-coupled device (CCD) chips with a resolution of 2048×2048 each. The chips are arranged in 5 rows with 6 in each row. The optical filters used in the camera setup have specific wavelengths and are represented by the labels u, g, r, i, and z. Each row observes the space through various optical filters (u, g, r, i, z) with different wavelengths $u = 354$ nm, $g = 475$ nm, $r = 622$ nm, $i = 763$ nm, and $z = 905$ nm, where nm stands for nano meters. Of these 5 filters, only the g and r filters lie in the visible part of the electromagnetic spectrum. These filters correspond to green and red colors respectively.

It is estimated that every night SDSS captures about 200 GB of data. I will be using data from the latest data release i.e., the SDSS DR18 (Data Release 18) (Almeida and et. al, 2023). These filters play a significant role in observing the space and capturing different aspects of the electromagnetic spectrum.

2.1.1 Telescope filters

- **Filter u (354 nm)**: This filter allows the observation of ultraviolet light, enabling the detection of objects or phenomena that emit or interact strongly with ultraviolet radiation.
- **Filter g (475 nm)**: This filter captures light in the blue-green part of the spectrum. It is particularly useful for studying objects that emit light in this range, such as certain types of stars and galaxies.
- **Filter r (622 nm)**: This filter is sensitive to red light, enabling the detection of objects that emit or reflect red wavelengths. It is often used to study the characteristics of red stars, galaxies, and certain astronomical phenomena.

- **Filter i (763 nm)**: This filter is designed to capture near-infrared light. It allows the observation of objects that emit or interact strongly with infrared radiation, providing insights into phenomena such as star formation, dust clouds, and certain celestial objects that emit primarily in the infrared.

- **Filter z (905 nm)**: This filter extends further into the infrared part of the spectrum. It enables the detection of objects that emit or interact strongly with longer-wavelength infrared radiation, providing valuable information about objects such as distant galaxies, quasars, and other infrared-emitting sources.

By using these specific optical filters with distinct wavelengths, astronomers can gather data from different regions of the electromagnetic spectrum, allowing for a comprehensive study of celestial objects and phenomena across a wide range of wavelengths. These optical filters play a crucial role in star-galaxy-quasar classification. By observing celestial objects through different filters, astronomers can gather information about their spectral properties and distinguish between different types of objects.

The filters u, g, r, i, and z capture light at specific wavelengths, allowing astronomers to study the characteristics and emission patterns of stars, galaxies, and quasars. For example, the u filter enables the detection of blue light, which is often associated with young, hot stars. The r and i filters are sensitive to red and near-infrared light, respectively, which can provide insights into the properties of red stars, galaxies, and objects emitting in the infrared spectrum. The z filter extends further into the infrared, enabling the observation of distant galaxies and quasars that emit predominantly in the longer-wavelength infrared range.

By analyzing the light captured through these filters and combining the observations with other techniques, astronomers can classify celestial objects as stars, galaxies, or quasars based on their distinct spectral signatures. This classification helps in understanding the nature and evolution of these objects, unraveling the mysteries of the universe.

2.2 Previous works

The development of big astronomical surveys has greatly increased the importance of automatic data classification and processing. The separation of photometric catalogs into stars and galaxies must be automated because there is simply too much data for human specialists to manually classify (Kim and Brunner, 2016). These large surveys are aimed at generating astronomical catalogues which then can be used for further studies of the objects. This makes accurate classification a crucial step.

There are 3 major categories of galaxies according to their morphologies as per Hubble galaxy classification scheme (Hubble, 1926). Any manual classification and analysis of large datasets requires a considerable amount of time and effort. The categorization of galaxies is difficult and imprecise due to the complicated nature of galaxies and the characteristics of the pictures (Khalifa et al., 2018). For scenarios with several categories, deep learning has shown notable results and significantly improved visual detection and identification. The Convolutional Neural Network (CNN) is the most prevalent

type of deep, feed-forward neural network and one of the most renowned deep learning algorithms (Khalifa et al., 2018). The term "feed-forward" refers to the flow of information through the network in a single direction, i.e., moving forward from the input layer to output layer without any feedback connections.

Instead of using image data from the astronomical surveys, recently, accurate stellar object classification using photometric data has gained much popularity (Wierzbiński et al., 2021). Machine Learning (ML) algorithms consists of the model parameters and hyper parameters. Hyperparameter is a configuration that is external to the model and whose value cannot be determined from the data. With increasing complexity of model, the number of hyperparameter configurations to be determined increases (Cui and Bai, 2019).

Clarke et al. (2020) trained a random forest machine learning model on 1.55 million astronomical sources using the SDSS DR15 dataset (3 versions old). They also used cross validation technique to tune the hyper-parameters in the model and to ensure that the class imbalance does not create a bias. The F1-scores for the classification model achieved by them were 0.99, 0.95 and 0.97 fro galaxies, quasars and stars respectively.

One of the current practices to set hyperparameters is manual search, that is basically relying on human intuition and experience. The hyperparameters that work on one set of data are not guaranteed to work on another set of data (Young et al., 2015). For hyperparameter tuning Clarke et al. (2020) used a technique called grid search. Grid search involves defining a grid of possible values for each hyperparameter that needs to be tuned. The method then systematically evaluates all possible combinations of hyperparameters from the grid by training and testing the model using cross validation. Grid search carries out exhaustive search in the hyperparameter space and can be computationally expensive and time consuming (Clarke et al., 2020). Another technique called Randomized Search follows the same idea as grid search, but instead of exhaustively trying out all possible combinations of hyperparameters, randomized search randomly selects sets of hyperparameter values untill highest classification accuracy is achieved. This can also get computationally expensive and time consuming, so a limit can be set as to how many combinations of hyperparameters can be explored.

2.3 Machine Learning

Machine learning is a subset of artificial intelligence (AI) that focuses on developing algorithms and techniques that enable computers to learn from data and improve over time without having to be explicitly programmed. Based on the data it has been exposed to, the machine is able to recognize patterns, make informed decisions, and make predictions.

At its core, machine learning makes use of mathematical and statistical models in order to enable computers to learn from experience. Programming traditionally entails giving explicit instructions to a computer, whereas machine learning involves the computer learning from examples and data in order to make predictions or decisions.

A machine learning algorithm can be classified into several types, each with its own characteristics and applications. Supervised learning involves training a model on labeled data, in which the algo-

rithm learns the relationship between input data and output labels. For example, this type of learning is used to categorize data (categorizing it into classes) and predict numerical values (using regression data). The two other types are unsupervised learning (deals with unlabelled data) and reinforcement learning where an agent learns by interacting with an environment to achieve certain goals.

For the purposes of this thesis we will focus on supervised learning methods as the dataset is labelled. Based on works by Wierzbiński et al. (2021) and Clarke et al. (2020) we will be focusing on Random Forest Classifier, Gradient Boosting Classifier and Logistic Regression for this analysis.

2.3.1 Random Forest Classifier

The ideology of integrating multiple methods or models to build a predictive model is called ensemble learning methodology(Rokach, 2010). Random forests, also known as random decision forests, are an ensemble learning method widely utilized in various domains, including classification and regression tasks. The technique involves constructing an ensemble of decision trees during the training phase.

In the context of classification tasks, random forests aggregate predictions from multiple decision trees to determine the final class assignment. Each decision tree within the forest independently evaluates a subset of features and a random subset of the training data. By incorporating these randomized components, the random forest fosters diversity among the individual trees, mitigating the risk of overfitting and enhancing robustness.

The underlying principle of random forests centers around the notion that aggregating the predictions of multiple decision trees can yield superior performance compared to any single tree. By combining the outputs of diverse decision trees, the ensemble model is capable of mitigating individual tree biases and errors, resulting in improved generalization and enhanced predictive power (Ho, 1995).

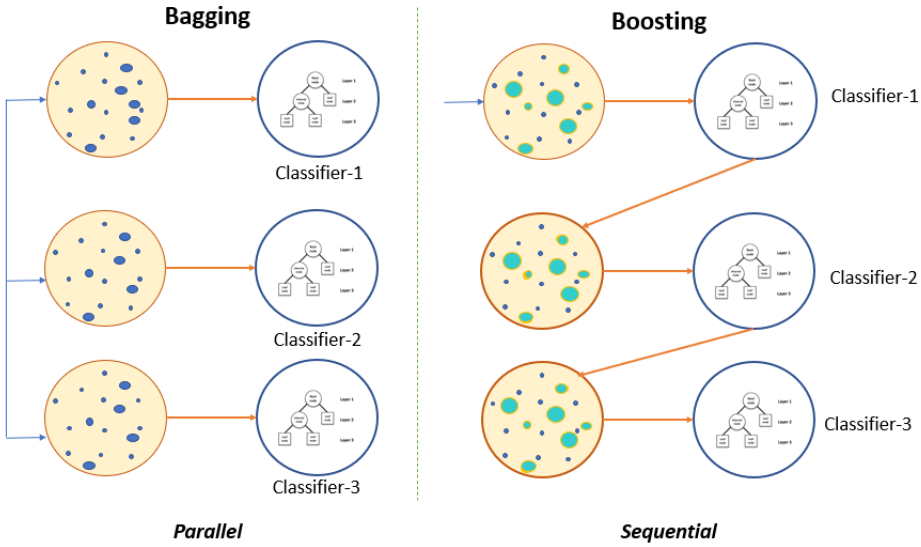


Figure 2.1: Bagging vs Boosting (Singhal, 2020)

Bagging (Bootstrap Aggregating) is an ensemble learning technique that uses the bootstrap sampling approach i.e., create multiple datasets and train a separate model on each dataset. These models

are then combined or aggregated to make predictions in the final ensemble. Bagging technique is commonly used to reduce the variance and improve the overall performance of machine learning model (Breiman, 2001). Bagging is employed for two primary purposes. Initially, it is observed that bagging contributes to improved accuracy, particularly when employing random features. Additionally, bagging serves the purpose of continually providing estimates for both the combined ensemble of trees' generalization error, as well as assessments for their strength and correlation (Breiman, 2001).

Due to these characteristics random forests have wide applicability in astronomical object classification (Becker et al., 2020).

The important hyperparameters in Random Decision Forests are (Pedregosa et al., 2011):

- **Number of trees**: Random Forest creates an ensemble of decision trees, and this hyperparameter determines the number of trees in the ensemble. A larger number of trees can improve the model's accuracy but increases the computational cost. If the trees are grown very deep, there is a tendency to learn irregular patterns and overfitting occurs. Random forests take average of multiple decision trees which are trained on different parts of training dataset. This helps in reducing the overall variance and improves the performance of the model.

- **Maximum depth**: This hyperparameter restricts the depth of each decision tree in the Random Forest. Controlling the maximum depth helps to prevent overfitting. A deeper tree can capture more complex relationships in the data, but it may also lead to overfitting if the depth is not properly constrained.

- **Number of features**: At each node of a decision tree in Random Forest, a random subset of features is considered for splitting. This hyperparameter determines the number of features to be considered. It provides randomness to the model and reduces the correlation between trees.

- **min_weight_fraction_leaf**: The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

2.3.2 Gradient Boost Classifier

Gradient boosting just like Random forests is another ensemble learning technique. The concept of gradient boosting comes from Leo Breiman's discovery that boosting can be thought of as a way to improve a function by making it better step by step (Breiman, 2001). The term boosting originates from the concept of sequentially boosting the performance of a weak learner by focusing on the examples it struggles with. In this context, "boost" means to increase or enhance, which reflects the algorithm's objective of improving the predictive power of weak learners (Rokach, 2010). The idea behind boosting is to iteratively train new weak learners, giving more attention to the instances that the ensemble struggles to classify correctly. By doing so, each subsequent weak learner "boosts" the performance of the overall ensemble by correcting the errors made by the previous learners. (Hastie et al., 2009*a*)

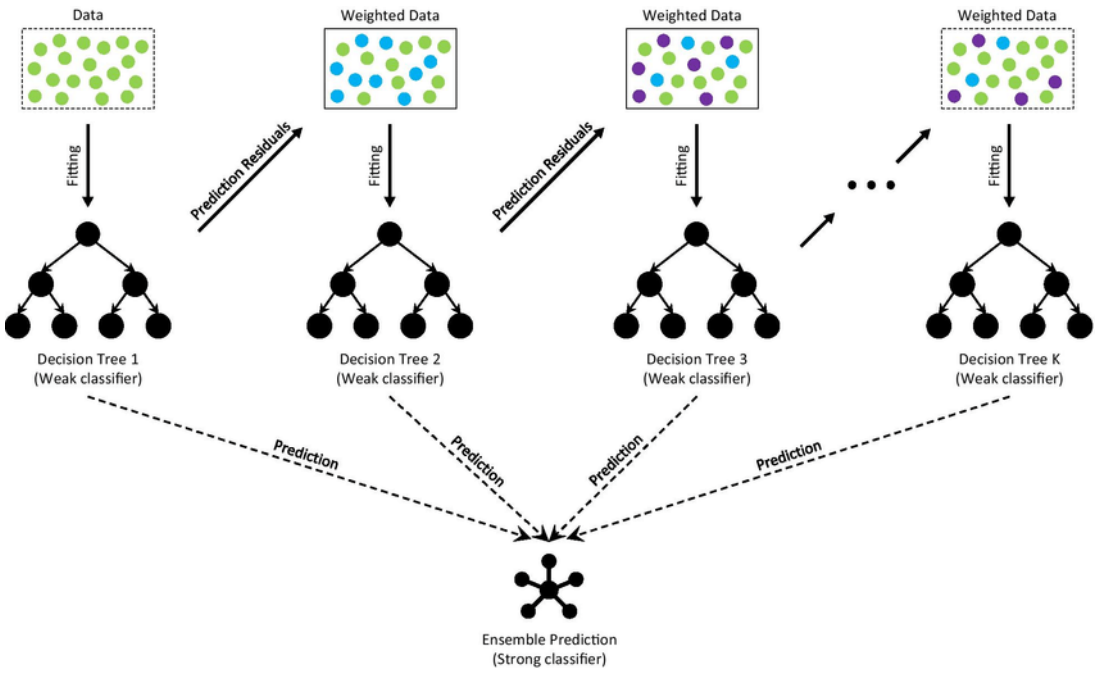


Figure 2.2: Gradient Boosting in a nutshell (Deng et al., 2021)

Gradient Boosting assigns weights to the residuals (differences between actual and predicted values) rather than to instances (see Figure 2.2). The algorithm aims to minimize these residuals using gradient descent, iteratively improving the predictions. Gradient Boosting uses various loss functions depending on the problem, such as mean squared error for regression and cross-entropy for classification. The algorithm optimizes these loss functions by minimizing the gradients, hence the name Gradient Boosting (Friedman, 2001). The gradient of a function at a specific point is a vector that points in the direction of the steepest increase of that function. In the context of optimization, the gradient points toward the direction in which the function's output increases the most.

Gradient boosting is working to improve the model's performance by adjusting its parameters in a way that reduces the loss function. To do this, the algorithm utilizes the gradients of the loss function with respect to the model's parameters. By iteratively updating the parameters in the opposite direction of the gradients, the algorithm aims to find the values that lead to the lowest possible loss, thereby improving the model's predictive accuracy.

The hyperparameters in Gradient Boosting Classifier are (Pedregosa et al., 2011):

- **n_estimators**: This parameter determines the number of weak learners(trees) to be sequentially added to the ensemble. A higher number generally improves model performance, but it also increases computational cost.

- **learning_rate**: The learning rate also known as shrinkage or step size, is the parameter that controls the contribution of each weak learner to the ensemble. A lower learning rate makes the algorithm more robust and requires more iterations to converge, but it can prevent overfitting. Conversely, a higher learning rate allows for faster convergence but may lead to overfitting if not controlled properly.

- **max_depth**: This sets the maximum depth of each weak learner (tree). It controls the complexity of the trees and affects their ability to capture complex relationships in the data.

These are a subset of hyperparameters available in a Gradient Boosting Classifier, refer the sklearn GradientBoostingClassifier (Buitinck et al., 2013) online documentation for a exhaustive list.

2.3.3 Logistic Regression

Regression techniques are essential in data analysis when you want to understand how one thing depends on others. Linear regression consists of modelling the relationship between a continuous dependent variable and one or more independent variable. The predicted value in linear regression can range from negative to positive infinity, making it suitable for predicting continuous outcomes (Hosmer and Lemeshow, 2000).

In logistic regression, the outcome variable takes on a binary or dichotomous form, meaning it assumes discrete values. In essence, logistic regression consists of getting the maximum likelihood of an observation belonging to a particular class within a binary classification framework. This technique can be expanded towards estimating probabilities for categorical outcomes, setting it apart from conventional linear regression techniques focusing on predicting continuous numerical values (Hosmer and Lemeshow, 2000).

The key differences between linear and logistic regression lies in the nature of outcome:

- **Linear Regression:**
  - **Outcome Variable:** Continuous numerical values.
  - **Range of Predicted Values:** Can span the entire real number line.
  - **Objective:** Minimize the squared differences between predicted and actual values.
- **Logistic Regression:**
  - **Outcome Variable:** Binary or categorical values (typically 0 or 1).
  - **Range of Predicted Values:** Restricted between 0 and 1, representing probabilities.
  - **Objective:** Minimize the log loss (cross-entropy) between predicted probabilities and actual classes.

The "log loss" (also known as cross-entropy loss) is a mathematical measure used to quantify the difference between the predicted probabilities and the actual classes. It assesses how well the predicted probabilities match the observed outcomes. The main idea is that the log loss is smaller when the predicted probabilities are close to the actual classes and larger when they deviate from the actual classes.

The hyperparameters in Logistic Regression are (Pedregosa et al., 2011):

- **Cs**: This controls the set of inverse regularization strengths (C values) that the algorithm will consider during cross-validation. Regularization applied to a model refers to the incorporation of a penalty term into the model's optimization process. This penalty term is added to the objective function that the model aims to optimize during training. The primary goal of regularization is to prevent the model from becoming overly complex by discouraging it from fitting the training data too closely. By doing so, regularization aims to improve the model's ability to generalize well to new, unseen data. Regularization helps strike a balance between bias and variance in the model. A model with high complexity (low regularization) can fit the training data very closely, potentially capturing noise in the data. This leads to low bias but high variance, making the model prone to overfitting. Regularization introduces a controlled amount of bias by penalizing complex models, which in turn reduces variance and improves generalization to new data.

- **penalty**: Determines the regularization to be applied in the model. The options are 'L1', 'L2', 'elasticnet', and 'None'. L1 regularisation encourages sparsity in the model's coefficients. L2 regularisation prevents large coefficients and reduces overfitting. Elasticnet is the combination of both L1 and L2. L1 regularization adds a penalty to the objective function that is proportional to the absolute values of the model's coefficients. Mathematically, it can be represented as the sum of the absolute values of the coefficients. L2 regularization adds a penalty to the objective function that is proportional to the squared values of the model's coefficients. L1 encourages sparsity and feature selection, L2 prevents large coefficients, and Elastic Net combines both techniques to provide a flexible approach that balances their advantages.

- **solver**: Determines the optimisation algorithm to be used to fit the logistic regression model. Some of the options are 'newton-cg', 'lbfgs' and 'sag' which are suitable for multinomial logistic regression. 'liblinear' is better for small datasets. In our case we will be using 'saga' (Stochastic Average Gradient descent with Adaptive regularisation) as it is versatile and can handle both L1 and L2 regularisation.

- **max_iter**: Maximum number of iterations for the optimisation algorithm to converge.

This is a subset of key hyperparameters in 'LogisticRegressionCV' module of sklearn package. For a full exhaustive list of hyperparameters refer to the official documentation of the LogisticRegressionCV module of the sklearn package (Buitinck et al., 2013).

2.3.4 Support Vector Machines (SVMs)

The support-vector network follows the following theory: it maps the input vectors into a high-dimensional feature space using a non-linear mapping that is predetermined in advance. A linear decision surface with unique characteristics is built in this area to ensure the network's excellent generalization ability (Cortes and Vapnik, 1995). Support Vector Machines (SVMs) is a supervised machine learning algorithm used to solve classification and regression problems. The core idea is to find

the hyperplane in the feature space to separate two classes (Jin et al., 2019). Like fitting a line in 2D to separate 2 classes, hyperplane separates multidimensional feature space. An optimal hyperplane, see Figure 2.3 can be defined as a linear decision function with maximum margin between the vectors of two classes (Cortes and Vapnik, 1995).

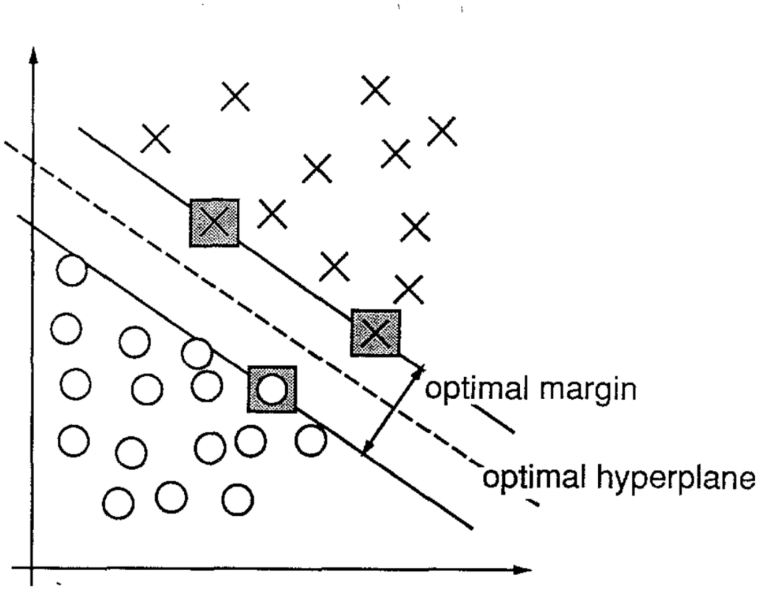


Figure 2.3: The support vectors, marked with grey squares, define the margin of largest separation between the two classes (Cortes and Vapnik, 1995).

One just has to account for a tiny portion of the training data, the so-called support vectors, which define this margin, in order to create such ideal hyperplanes (Cortes and Vapnik, 1995). SVM has been used widely for classification in astronomy (Zhang et al., 2012) (Kim and Brunner, 2016). In SVM, the hyperparameters are, Kernel, C and Gamma (Buitinck et al., 2013).

- **Kernel**: SVM can use different kernel functions to transform the input data into a higher-dimensional space. The kernel function determines the type of decision boundary created by SVM, such as linear, polynomial, or radial basis function (RBF) kernels.
- **C**: This hyperparameter controls the trade-off between maximizing the margin (distance between the decision boundary and the nearest data points) and minimizing the classification errors. A smaller value of C allows more misclassifications but increases the margin, while a larger value of C reduces the margin but minimizes misclassifications.
- **Gamma**: This hyperparameter is specific to the RBF kernel and determines the influence of each training example. A higher value of gamma leads to a more complex decision boundary that can fit the training data more precisely. However, it may also result in overfitting if the value is too large.

2.3.5 Cross Validation

Cross validation is a re-sampling technique which involves partitioning the dataset into subsets (folds), training the model on some folds and evaluating performance on the remaining folds. Cross validation helps to obtain a better estimate of the model's generalisation on new/unseen data. In the K-fold cross validation technique, the dataset is divided into K subsets of roughly equal size. The model is trained K times, each time using K-1 folds for training and 1 fold for validation. The validation fold is then rotated so that every fold gets a chance to serve as the validation set. The performance metrics obtained from each fold are averaged to get an overall assessment of the model's performance (Hastie et al., 2009*b*).

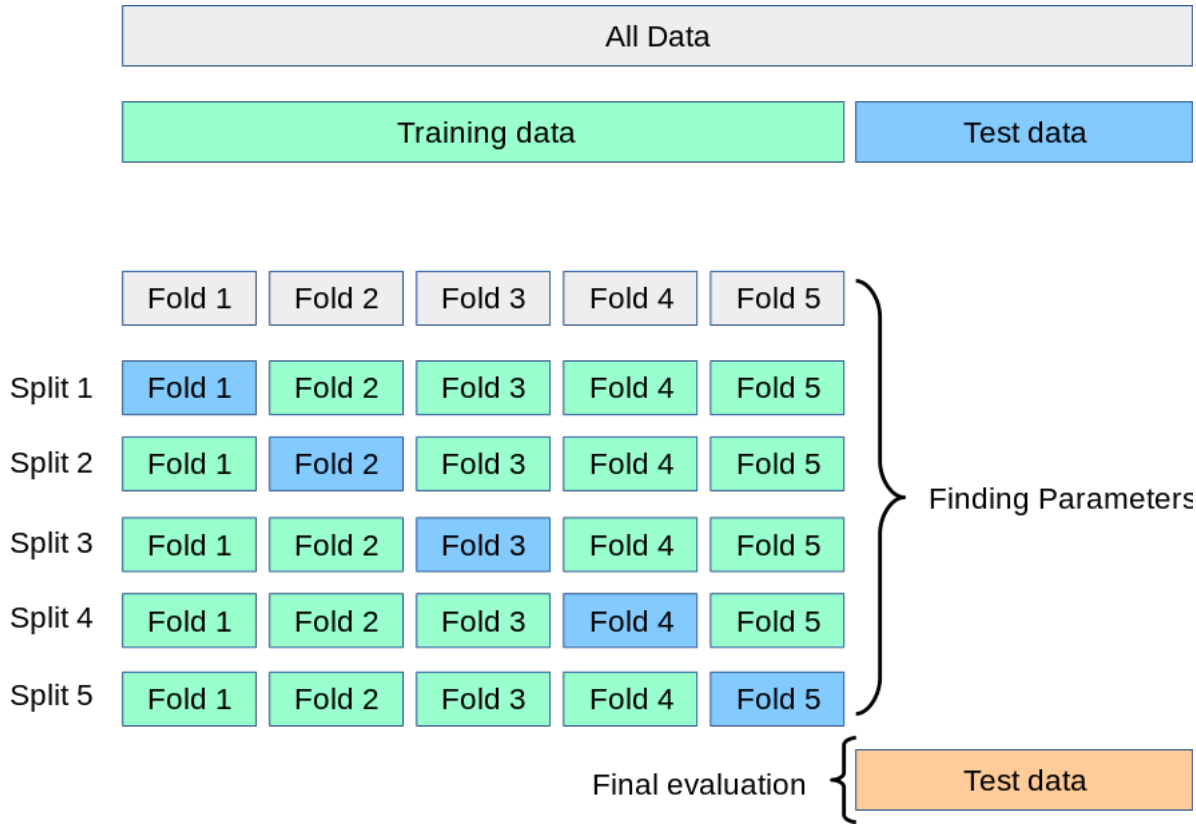


Figure 2.4: Example of 5 fold cross validation (K = 5) (Scikit-learn contributors, 2023)

In certain classification scenarios, there's a possibility that the distribution of target classes might be significantly imbalanced. For instance, one class could have far more instances compared to another class. Figure 4.1 shows the class imbalance in the dataset used for this thesis. To address this situation, it's advisable to employ stratified sampling techniques like those provided by StratifiedKFold and StratifiedShuffleSplit. These techniques help maintain the proportional representation of classes in both training and validation sets, ensuring that the relative frequencies of different classes are preserved across the folds (Scikit-learn contributors, 2023). This is important to prevent one class from being underrepresented in the validation set, which could lead to biased evaluation.

2.3.6 Overfitting and Underfitting

In machine learning, overfitting and underfitting are two common issues that can lead to poor performance of models on new/unseen data. These issues occur when the model is not capable of generalising from the training data.

Overfitting occurs when a model learns data too well which leads to learning underlying patterns as well as noise. This leads to model becoming too complex and fit the training data perfectly but fails to generalise new data. The telltale sign of overfitting is high accuracy on training data but poor performance on validation or test data (Hawkins, 2004).

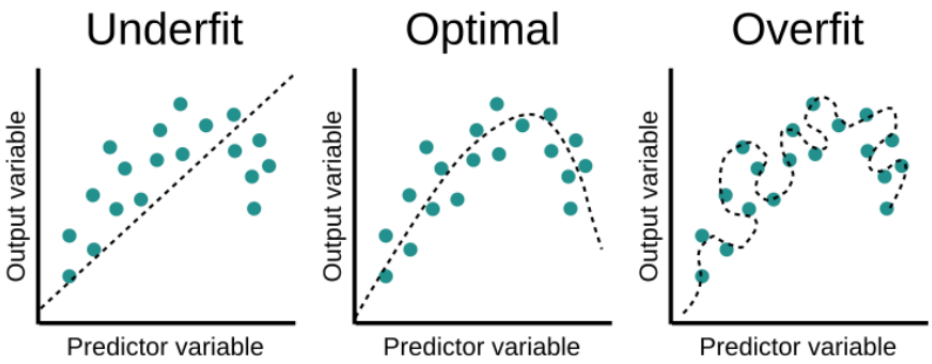


Figure 2.5: Graphical representation of overfit, underfit and good fit (fast.ai contributors, 2023)

Underfitting is when a model is unable to adequately learn the training data as well as struggles to make predictions on new data. An underfit model doesn't capture the complexity of the data and misses out on important details and relationships. It may be characterized by low accuracy on both the training and validation/test sets (Hawkins, 2004).

The goal in machine learning is to achieve a balanced fit, where the model captures the underlying patterns without being overly influenced by noise. A balanced fit generalizes well to new data, making accurate predictions. To avoid overfitting we can use techniques like regularisation that penalize overly large coefficients which helps of control model complexity. Using cross validation also helps mitigating overfitting. Sometimes gathering more data or increasing the size of data using data augmentation techniques can help the model generalize better. Finding the right balance between model complexity and generalization is a fundamental challenge in machine learning, and addressing overfitting and underfitting is crucial for building effective predictive models.

2.3.7 Machine Learning Metrics

The outputs of machine learning algorithms require careful assessment and this analysis must be interpreted correctly. For this purpose a set of metrics are used to assess the performance of machine learning models. Many metrics are available; however, we will focus on the ones that are commonly used and available in most machine learning frameworks.

Confusion Matrix

A confusion matrix is a tabular representation that helps visualize the performance of a classification model by comparing predicted class labels to actual class labels. It's particularly useful for understanding how well the model is doing in terms of true positives, true negatives, false positives, and false negatives (Tharwat, 2021).

- **True Positives (TP):** The number of instances that were correctly predicted as positive (correctly classified as the positive class).
- **True Negatives (TN):** The number of instances that were correctly predicted as negative (correctly classified as the negative class).
- **False Positives (FP):** The number of instances that were incorrectly predicted as positive (incorrectly classified as the positive class when they actually belong to the negative class). Also known as a "Type I error."
- **False Negatives (FN):** The number of instances that were incorrectly predicted as negative (incorrectly classified as the negative class when they actually belong to the positive class). Also known as a "Type II error."

		Predicted	
		Positive	Negative
Actual	Positive	True Positives	False Negatives
	Negative	False Positives	True Negatives

Table 2.1: Confusion Matrix

Accuracy

Accuracy is defined as the ratio of correct predictions to the total number of predictions. It is the most common metric used to evaluate performance of a machine learning algorithm. It provides a general overview of model's correctness but it is not robust as imbalanced classes can give misleading accuracy scores (Tharwat, 2021).

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Precision

Precision also referred to as *Positive prediction value (PPV)*, is defined as the ratio of true positive predictions to the total number of positive predictions. This metric is also affected if the data is imbalanced (Powers, 2011).

$$PPV = Precision = \frac{TP}{FP + TP} \tag{2.2}$$

Recall

Recall also referred to as *True positive rate (TPR)* or *Sensitivity* is defined as the ratio of true positive predictions to the total number of actual positives. This metric is relevant when false negatives predictions are risky especially in the field of medicine. High recall indicates that the model is capturing most positive instances correctly (Tharwat, 2021) (Powers, 2011).

$$Recall = \frac{TP}{FN + TP} \tag{2.3}$$

F1 Score

The F1-score, often used in machine learning for classification tasks, is defined as the harmonic mean of precision and recall. In mathematics, the harmonic mean is a type of average that's particularly useful when dealing with rates or ratios. F1-score ranges from zero to one and high value can be interpreted as good classification performance (Powers, 2011).

Harmonic mean is calculated by taking the reciprocal of the arithmetic mean of the reciprocals of a set of values. The formula for the harmonic mean of values $x_1, x_2, x_3 ... x_n$ is:

$$H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} \tag{2.4}$$

Using equations for precision and recall - Equation 2.2 and Equation 2.3 we can write the equation for F1-score using the equation for Harmonic mean Equation 2.4:

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{2TP}{2TP + FP + FN} \tag{2.5}$$

F1-score balances precision and recall, giving a single score that considers both false positives and false negatives (Tharwat, 2021).

Logloss

Logarithmic Loss, commonly known as Log Loss or Cross-Entropy Loss, is a widely used evaluation metric for classification problems, particularly in scenarios where the model's predicted probabilities are important. It measures the performance of a classification model by quantifying the difference between predicted probabilities and actual class labels. Log loss values range from 0 to positive infinity with lower values indicating better model performance. Log loss heavily penalizes incorrect high-confidence predictions. If a model assigns high probabilities to incorrect classes, the log loss will increase substantially (Buitinck et al., 2013).

2.4 Genetic Algorithms (GAs)

According to Darwin's rule of natural selection, organisms with phenotypes that are well matched to their immediate environment will eventually have a higher probability of reproducing and/or surviv-

ing than other, less suitable species (Dodson, 1976). Phenotype refer to the observable physical and behavioral characteristics of an organism, resulting from the interaction between its genetic makeup (genotype) and its environment. In the following generations, only those phenotypes prevails which are more suited to the environment. Mutation of genes can cause the organism to either have better characteristics or worse characteristics. But natural selection will eventually carry forward those mutations which are better.

Genetic Algorithm (GA) is a computational model which takes inspiration from Darwin's law of natural selection. The terminologies used to describe such types of algorithms are analogous to the biological ones. A population-based search technique called the Genetic Algorithm (GA) creates new populations by repeatedly using genetic operators or randomization functions (Katoch et al., 2021). GA effectively searches an encoded parameter space by using the objects from the population and the randomization technique (Liu, 2019).

A population is initialized using random variables (traits/characteristics), a set of variables is referred to as chromosomes. The main components of GA are chromosomal representation, crossover, fitness selection, mutation, and fitness function computing. Until off-springs generated provide the ideal solution or the maximum number of iterations is achieved, the selection, crossover, and mutation procedures on the population are repeated. GA can be summarised in the form of a pseudo-code.

The fundamental theorem of GA is the Schema Theorem (Liu, 2019). "According to Schema theorem, the original schema has to be replaced with modified schema. To maintain the diversity in population, the new schema keeps the initial population during the early stages of evolution. At the end of evolution, the appropriate schema will be produced to prevent any distortion of excellent genetic schema." (Katoch et al., 2021) The most crucial element of genetic operators is the crossover operation. It is a recombination operator that joins pieces of the chromosomes of two parents to create offspring that have a portion of both parents' genetic makeup. (Tang et al., 1996). "The offspring generation formed by the cross-interchange of the parental chromosomes is likely to discard or destroy the excellent genetic schemas possessed by the parental individual."(Liu, 2019)

2.4.1 Genetic Operators

Genetic operators are the operators that are used during the optimisation process. See Figure 2.6 the general operators are encoding schemes, selection, crossover and mutation.

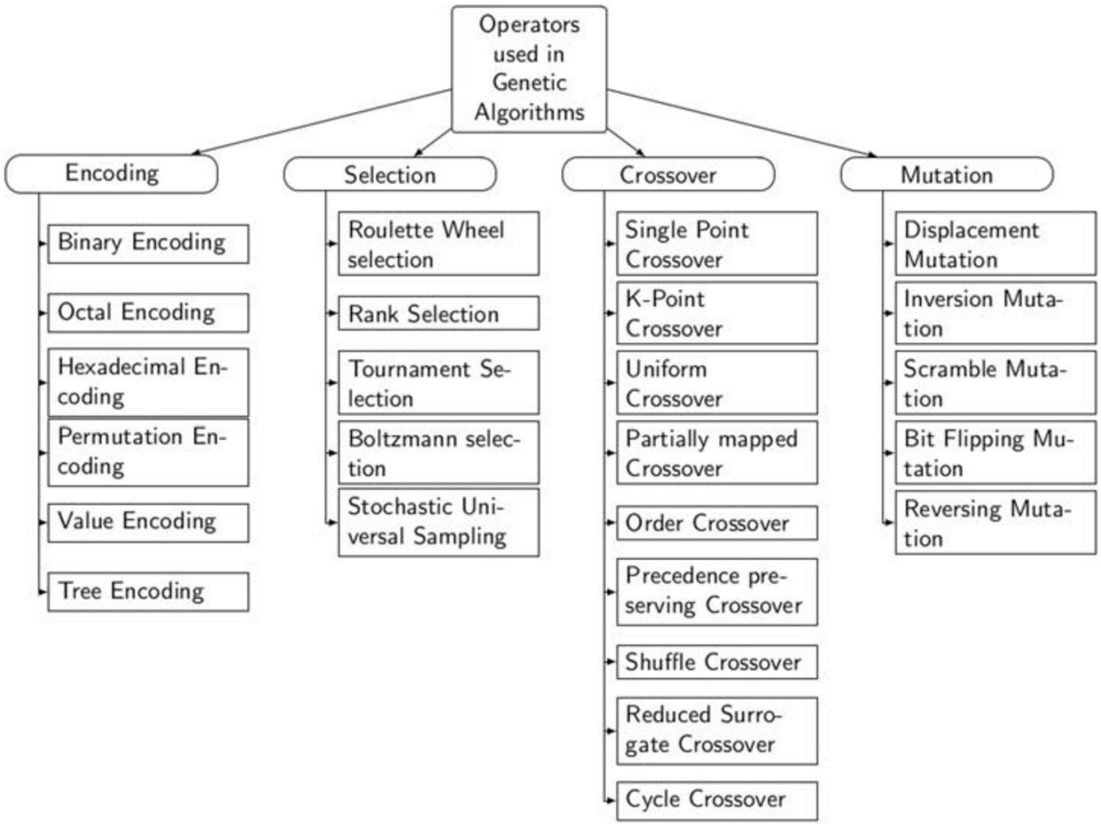


Figure 2.6: Operators used in Genetic Algorithms (Katoch et al., 2021)

Representation (Encoding Schemes)

We encode the solution set of the problem to improve the algorithm's performance. This encoded object functions analogous to a chromosome. It is not the problem in itself, but rather a collection of attempted solutions to the problem. The chromosome can be represented / encoded as binary, octadecimal, or hexadecimal values, and these can then be subjected to crossover and mutation to provide a more likely set of solutions. Aside from numerical encoding, "permutation encoding" is generally preferred in ordering problems.

Selection

In genetic algorithms, selection is a key phase that determines whether a particular string participates in reproduction. Boltzmann, rank, tournaments, roulette wheels, and universal probabilistic sampling are examples of well-known selection techniques. The roulette wheel selection method maps all possible strings to the wheel and assigns each string a space on the wheel based on the fit value. This wheel is then randomly rotated to select a specific solution involved in building the next generation (Holland, 1992) (Katoch et al., 2021).

The modified version of roulette wheel selection is ranked selection. Instead of using fitness value, it uses rankings. Brindle introduced the tournament selection approach in 1983, in which the individuals are chosen in pairs from a stochastic roulette wheel based on their fitness scores. Following selection, those with a greater fitness value are added to the pool of the following generation. (Holland, 1992)

The existing roulette wheel selection method is extended by stochastic universal sampling (SUS). It chooses an individual at regularly spaced intervals from a list of individuals from a generation, starting at a random position. All individuals have an equal chance of being chosen to take part in the following generation's crossover (Katoch et al., 2021).

Crossover

In order to create offspring, crossover operators combine the genetic information of two or more parents.

Many GA practitioners believe that the crossover operator is what really sets the GA apart from all other optimization algorithms. There are several crossover operation variants offered, with a single-point crossover being the most basic. Based on the selection methodology, the parents are randomly chosen. The regions of the two chromosomes beyond the crossover point are exchanged to create the offspring, where the crossover point is determined by a random process (Tang et al., 1996).

Similar to single-point crossover, multipoint crossover uses m crossover sites that are randomly selected without duplication. The crossover operator that is most often employed is partially matched crossover (PMX). This operator outperforms the majority of the other crossover operators in terms of performance (Katoch et al., 2021).

Mutation

An operator that modifies the chromosome is called mutation. The mutation can take place both on a global and local level. The procedure randomly changes the value of a string position occasionally (and typically with low probability p) (Tang et al., 1996). The three most well-known mutation operators are scramble mutation, simple inver-sion, and displacement.

The scramble mutation (SM) operator arranges the components of each individual solution in a random order and determines whether the freshly created solution's fitness value has increased or decreased (Jebari, 2013).

The SIM (simple inversion mutation operator) inversion operator flips the randomly chosen string and positions it in a random spot. Between any two given places in a single solution, the substring is reversed using the SIM (Jebari, 2013).

The displacement mutation (DM) operator moves an individual solution's substring within itself. To ensure that both the final solution and a random displacement mutation are legitimate, the location is randomly selected from the substring that is being used for displacement. Exchange mutation and insertion mutation are two types of DM variations. A portion of a unique solution is either swapped with another portion or inserted in a different position in exchange mutation and insertion mutation operators, respectively. (Jebari, 2013)

2.5 Tools and Technologies

This section will include all the tools and technologies that I used for the thesis with a breif description. The choice of programming language is Python due to its extensive documentation, open source nature and availability of sophisticated packages. For the workflow the choice of platform was Jupyter notebook (formerly IPython Notebook) which is an interactive way to write blockwise code. Due to its versatile nature it is a great tool for experimentation. Here are some key features of Jupyter notebooks:

- Interactive Execution: Jupyter Notebooks allow you to execute code cells interactively. Each code cell can be executed individually, and the results are displayed directly below the cell. This interactive nature enables users to experiment with code, test hypotheses, and observe immediate outcomes.
- Mixed Content: Notebooks support not only code but also markdown cells for rich-text formatting, LaTeXequations, images, links, and explanatory text. This combination of code and narrative makes it an effective tool for presenting and documenting analysis processes and findings.
- Visualization: Notebooks can display visualizations, graphs, and charts inline with the code, enhancing the ability to understand data patterns and relationships.
- Step-by-Step Exploration: Jupyter Notebooks encourage a step-by-step exploration of data and algorithms. By breaking down tasks into smaller blocks of code and text, it becomes easier to understand the logic behind each step.
- Reproducibility: Notebooks promote reproducible research by documenting code and results in a single document. This enables others to reproduce your work and verify your findings.
- Interactivity: Besides code execution, notebooks can incorporate interactive widgets and dashboards using libraries like ipywidgets. This enables dynamic manipulation of parameters and variables, enhancing the user's engagement with the content.
- Sharing and Collaboration: Jupyter Notebooks can be shared easily by exporting them to formats like HTML, PDF, and slides. They can also be hosted on platforms like GitHub, allowing others to view and interact with your analysis.
- Kernel Support: Jupyter Notebook's kernel architecture allows you to connect to different programming languages. This enables users to work with multiple languages within a single notebook.
- Educational Tool: Jupyter Notebooks are extensively used in educational settings to teach programming, data analysis, and scientific computing. The combination of code, explanations, and visualizations makes learning concepts more engaging.

Some of the Python libraries that were used for the data analysis and machine learning are:

- **NumPy**: Used in data analysis for numerical python which uses N dimensional array objects for mathematical operations (Harris et al., 2020).
- **Pandas**: Used mainly for working with relational or labelled datasets. It is based on NumPy increasing its efficiency and making it suitable for statistical analysis (Wes McKinney, 2010).
- **Matplotlib**: The primary plotting library in python with extensive community support (Hunter, 2007).
- **sklearn**: Machine learning library which provides all the necessary functionality right from preprocessing of data to evaluation of machine learning model performance. Detailed documentation with good community support makes it a very useful library for machine learning applications (Pedregosa et al., 2011) (Buitinck et al., 2013).
- **sklearn-genetic-opt**: Sklearn-genetic-opt uses evolutionary algorithms from the deap package to choose a set of hyperparameters that optimizes (max or min) the cross-validation scores, it can be used for both regression and classification problems.

For keeping track of research papers and citations I used Mendeley reference manager (Hennings, 2013). It is a tool that can extract relevant metadata - authors, title, year, journal, volume, issue etc using the DOI (Data Object Identifier) number of published research papers, conference proceedings, books, reports, journal articles, etc. The reference manager allows to read paper, make notes, highlight text in research papers and many other features useful for literature review. Mendeley reference manager also allows you to create a collection of research papers and export the bibliography file with all the metadata in multiple formats. LaTeXwas the choice of type setting format for writing the thesis as it is the best in class used to produce publication quality documents. LaTeX(pronounced ”lay-tech” or ”lah-tech”) is a typesetting system commonly used for creating documents with complex formatting, such as academic papers, research articles, theses, reports, books, and presentations. It is particularly popular in the fields of mathematics, computer science, engineering, and academia in general. LaTeX allows you to produce high-quality documents with consistent and professional-looking layouts, equations, tables, and references. The exported .bib file from mendeley can be used to add citations easily while writing the document. Final document was written on overleaf which is a cloud LaTeXcompiler as it is safe to have these files on cloud with version control.

Chapter 3

Methodology

This chapter will discuss on the process that was followed during the research i.e., the road-map for the research. The overall research design is experimental. Final aim is to determine the performance of using genetic algorithms to tune hyperparameters of the model and if we are able to obtain better metrics than default hyperparameters. Genetic algorithms can become computationally expensive if the hyperparameter space is large so some trial and error might be required to determine the feasibility of running genetic algorithm on large hyperparameter space.

3.1 Data Collection

The first step is to obtain the dataset from the skyserver service provided by the SDSS consortium (Almeida and et. al, 2023). The data is publicly available through the CAS and skyserver service provide by the SDSS consortium. Through the SQL search service (SDSS.org, 2022), I downloaded the top 500 thousand results with columns and filters recommended by SDSS.

```
SELECT TOP 500000
p.objid,p.ra,p.dec,p.u,p.g,p.r,p.i,p.z,
p.run, p.rerun, p.camcol, p.field,
s.specobjid, s.class, s.z as redshift,
s.plate, s.mjd, s.fiberid
FROM PhotoObj AS p
JOIN SpecObj AS s ON s.bestobjid = p.objid
WHERE
  p.u BETWEEN 0 AND 19.6
  AND g BETWEEN 0 AND 20
```

Listing 3.1: SQL query to download data from SDSS Skyserver

The TOP 500k results were taken so that the dataset can reflect the real distribution count of astronomical objects. The class imbalance in the dataset orginates from the fact that there are approximately 4 times more galaxies than stars and 6 times more than quasars (Clarke et al., 2020). The dataset is available in multiple formats, of which *.csv* was chosen for the ease of importing it as Pandas dataframe object in Python using the Pandas library.

3.2 Preprocessing and feature selection

The preprocessing of data will require some cleaning, handling missing values if any, feature normalisation or scaling using standard functions like StandardNormalScaler or MinMaxScaler from sklearn package. StandardScaler standardizes features by removing the mean and scaling them to unit variance. This means that after applying StandardScaler, the transformed features will have a mean of 0 and a standard deviation of 1. This scaling can be particularly useful when features have different units or different scales. MinMaxScaler scales features to a specific range, usually between 0 and 1. It linearly transforms the feature values such that the minimum value becomes 0 and the maximum value becomes 1. It's particularly useful when you want to maintain the original distribution of the data but scale it within a certain range. Principal Component Analysis (PCA) can also be performed to reduce the feature space and increase training speeds.

The next step is to split the dataset into training, testing and validation sets, ensuring that the class distribution is maintained in all sets. For this purpose a custom function will be created which utilises the *train_test_split( )* function from sklearn library. The *train_test_split( )* function takes split ratio as it's input parameter and splits the dataset into two parts. The custom function will take in 2 values of split ratios and split the dataset into 3 parts namely training set, testing set and validation set. In our case, of the total 500k rows, 300k will be used for training, 100k will be used for testing and the other 100k will be unseen data by the model which will be used for validation.

The selection of features can be based on a correlation matrix. There is a distinct spectral signature associated with each class of astronomical object. In the SDSS dataset, the u, g, r, i, and z columns represent different filters that are used to observe astronomical objects at specific wavelength ranges. The values in these columns correspond to the magnitudes of the objects in those bands. In telescopes, magnitude refers to a logarithmic measurement of brightness within a given band, and it is used to quantify the amount of light that reaches the telescope through each of its filters. Therefore, we can state that the values of these columns originate from the physical properties of astronomical objects, hence, they are correlated with the object class.

3.3 Establish baseline performance

Before implementing genetic algorithm to optimise hyperparameters of models it is important to establish the baseline performance of the models using the default hyperparameters. For this thesis, I have selected 3 machine learning models which are widely used in machine learning classification tasks - Random Forest Classifiers, Gradient Boosting Classifiers and Logistic Regression Classifiers. These models were selected based on work done by Wierzbiński et al. (2021) and Clarke et al. (2020).

Using the default parameters the models will be trained and the performance will be evaluated using the standard evaluation metrics such as accuracy, precision, recall and F1-score. These metrics will be saved for comparison with models trained after optimising hyperparameters.

3.4 Implementing Genetic Algorithms

Sklearn has a module named *sklearn-genetic-opt* which will be used to implement genetic algorithms in tuning hyperparameters. The functions from this library seamlessly integrate with machine learning models in sklearn library.

1. Define the Hyperparameter Space: Determine the range or set of possible values for each hyperparameter that needs to be tuned in scikit-learn algorithms. It is important to set a hyperparameter space because of limited computing resources.

2. Define the Fitness Function: Create a fitness function that evaluates the performance of a model with a specific set of hyperparameters. The fitness function should interface with scikit-learn by training and evaluating the model on the desired dataset. We will be using accuracy score as our fitness function.

3. Set Up the Genetic Algorithm: Use sklearn-genetic-opt to set up the genetic algorithm by defining the population, genetic operators (crossover and mutation), selection mechanisms, and termination criteria.

4. Integrate with scikit-learn: Within the genetic algorithm framework, sklearn will be used to train and evaluate the models with different hyperparameter settings based on the genetic algorithm's suggestions.

5. Retrieve the Best Hyperparameters: Once the genetic algorithm completes, the best hyperparameters can be extracted from the fittest individual(s) and then can be used to train the final model on the full dataset.

3.5 Evaluate results

Using the metrics from models trained using default hyperparamters and genetically optimised hyperparameters, draw conclusions as to which technique works best for our purpose.

Chapter 4

Observations and findings

This chapter consists of results of Python code that implemented genetic algorithm. The code can be found at a dedicated github repository (Barve, 2023) created for this thesis. Here's the link for the repository:

https://github.com/iamstarstuff/MScDataScienceThesis

The 'EDA.ipynb' Jupyter notebook file has the exploratory data analysis code and 'Final Compiled.ipynb' notebook has the machine learning models and genetic algorithm implementation.

The data downloaded from the SDSS SQL query server file name 'Skyserver_SQL5_24_2023 12_41_33 PM.csv' is also available on the repository.

4.1 EDA

Initial exploratory data analysis (EDA) was performed to gain deeper insights into the dataset. The dataset is of the shape (500000,18) i.e., 500 thousand rows and 18 columns. Of these 18 columns most of the columns are essentially metadata associated with each object. There are three unique classes of objects in our dataset which can be seen from *'class'* column. It is important to know the count of each class to determine if our dataset is balanced or imbalanced. Figure 4.1 shows that the dataset is balanced with more than half the objects belonging to the class 'GALAXY', about 39% of the objects belong to the class 'STAR' and 11% belong to the class 'QSO'. QSO stands for Quasi-Stellar objects, also known as Quasars. This shows clear imbalance of classes and this should be taken into account during training in order to ensure that our machine learning model takes it into consideration and learns all classes proportionally. This can be done using the StratifiedKFold cross validation technique, refer Figure 2.4.

Figure 4.2 shows the distribution of class of astronomical objects in the sky. Interestingly there are cluster of observations when only stars were observed. One other peculiar thing we can observe is how the observations are distributed in 2 seperated hemishpheres. The SDSS telescope, like many other ground-based telescopes, has limitations when it comes to observing objects at the zenith, which is the point directly overhead in the sky. This limitation is primarily due to the design and mechanics of the telescope, as well as the Earth's atmosphere. Telescopes are complex instruments with mechanical systems for pointing and tracking celestial objects. Some telescopes may have limitations in their

mechanical range of motion, preventing them from pointing directly overhead.

Table 4.1: SDSS DR18 Columns and Description.

Column Name	**Description**
objid	The unique identifier for the object in the PhotoObj table.
ra	The right ascension of the object in degrees.
dec	The declination of the object in degrees.
u, g, r, i, z	The magnitudes of the object in different bands (ultraviolet, green, red, near-infrared, and infrared).
run	The specific run in which the object was observed.
rerun	The rerun number for the run.
camcol	The camera column number.
field	The field number.
specobjid	The unique identifier for the object in the SpecObj table.
class	The classification of the object (e.g., star, galaxy, quasar).
z as redshift	The redshift value of the object, indicating its recession velocity.
plate	The plate number on which the object was observed.
mjd	The Modified Julian Date (MJD) of the observation.
fiberid	The fiber identification number.

The data required no cleaning as it has no null values or infinities that could cause problems with training. There were no missing values as well in the dataset which was verified with the pandas functions for these operations. Correlation matrix is a great way to identify potential features, so only those columns with higher correlation than other columns were selected.

Of these columns the correlation between class of object is highest with columns like the color bands u, g, r, i, z and redshift. Due to this only these columns were selected as features for training.

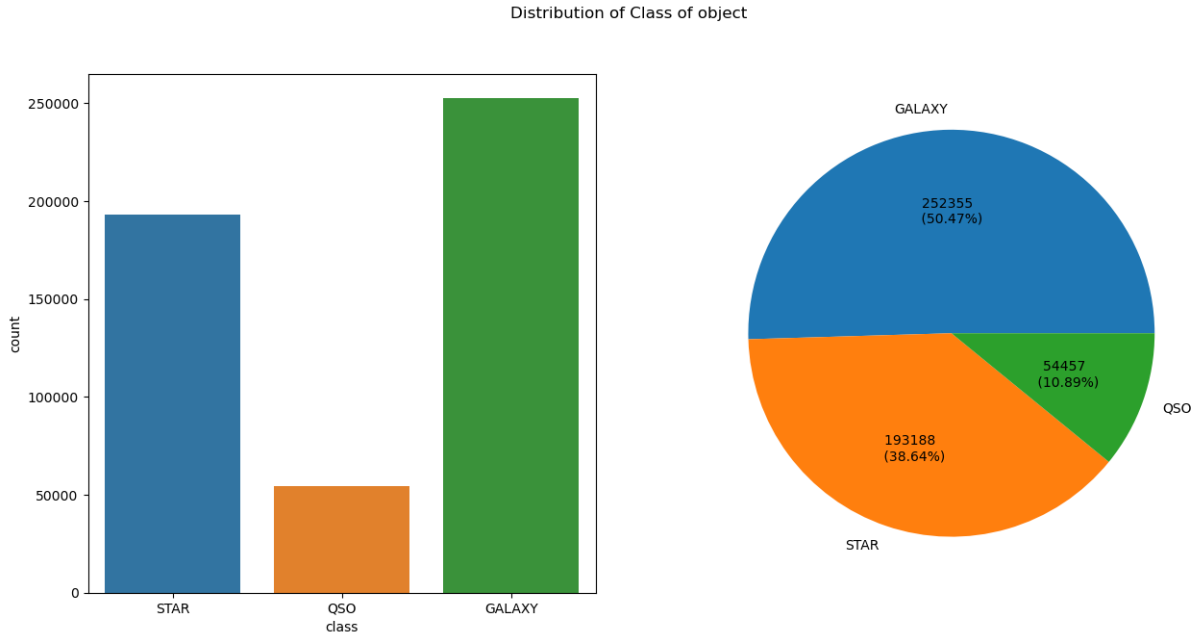


Figure 4.1: Distribution of Class of objects

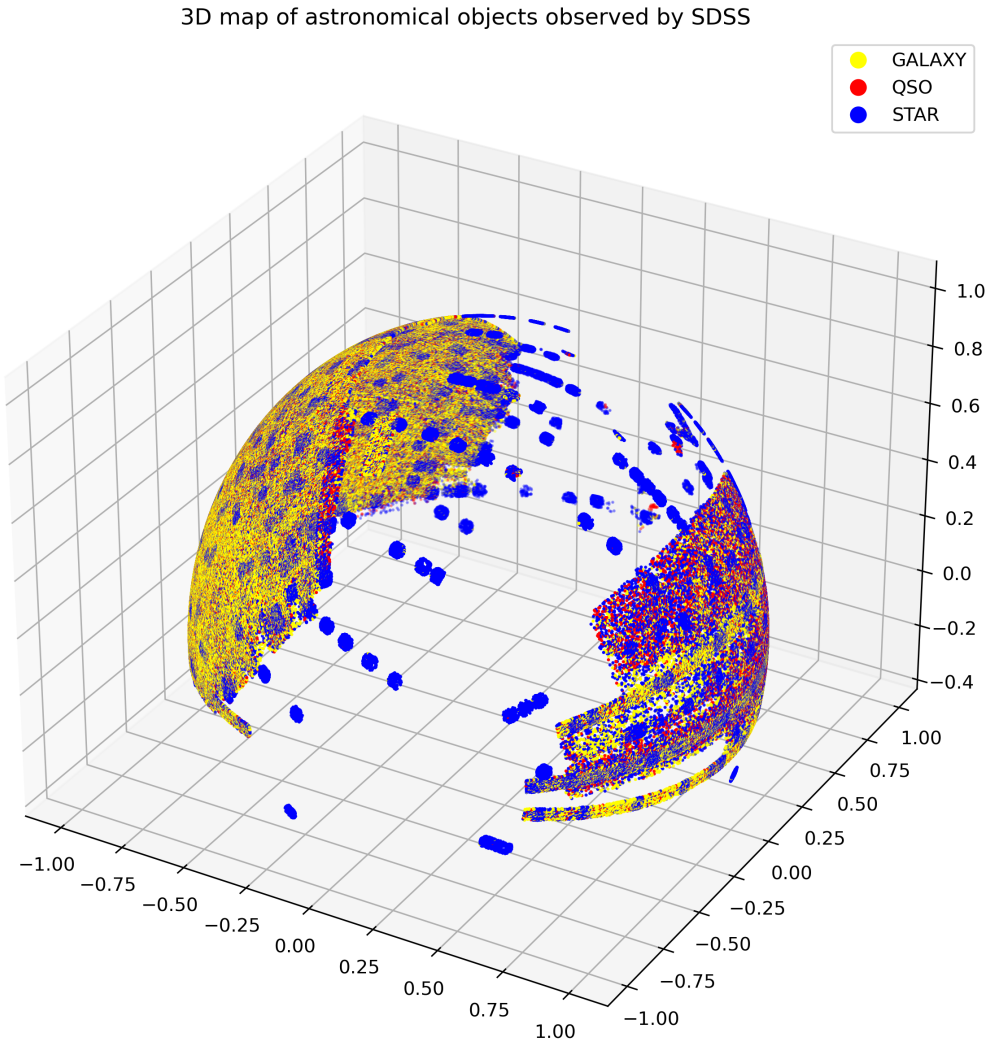


Figure 4.2: 3D Map of astronomical objects based on coordinates

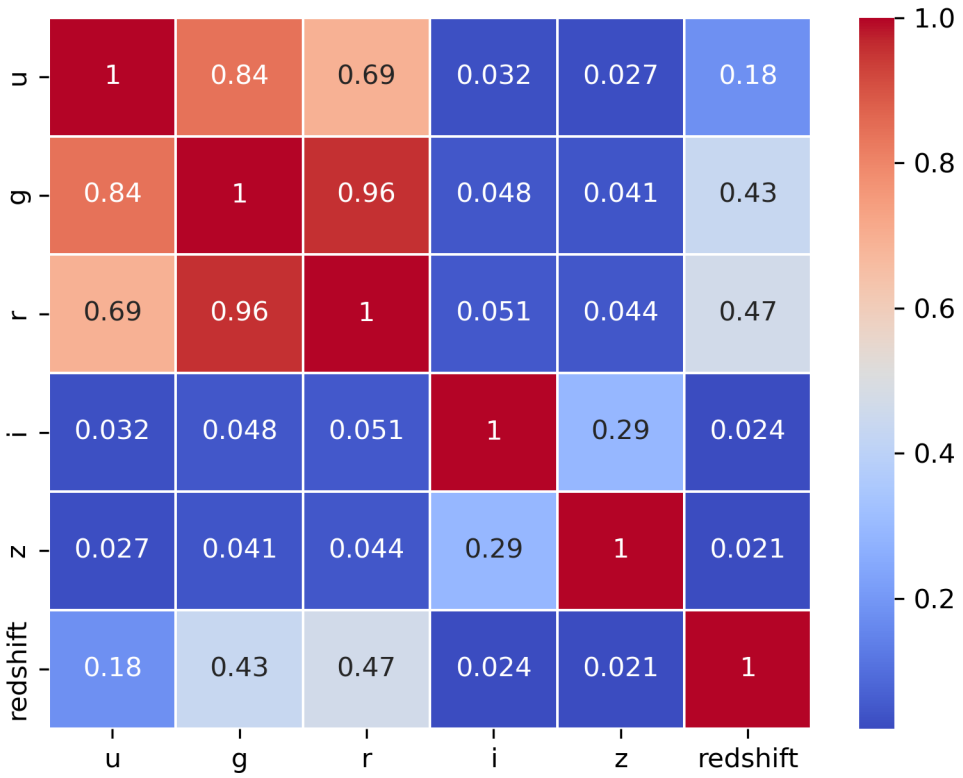


Figure 4.3: Correlation matrix of the color filters and redshift.

In the same way that I wrote a custom function to split data, I wrote a custom function to display evaluation metrics for the models. These function were heavily inspired from work done by Saiffudin (2022). The function plots confusion matrix for precision scores and recall scores, prints classification report and also prints logloss values. This enables quick report of model performance with all the necessary metrics required.

4.2 Model Performance

In this section, we will go through performance metrics of the models and also compare the performance after genetic optimisation.

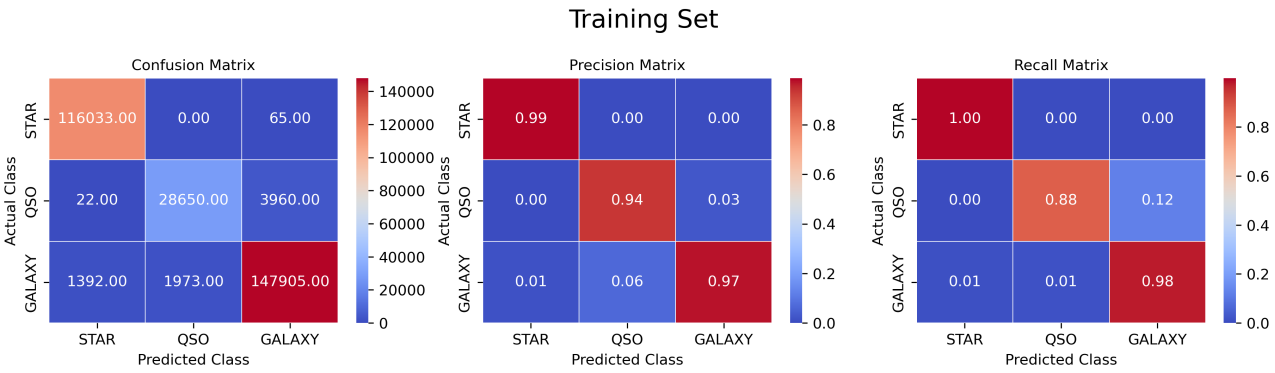
4.2.1 Random Forest Classifier

To establish baseline performance of Random Forest classifier, the model was first trained using the default hyperparamters.

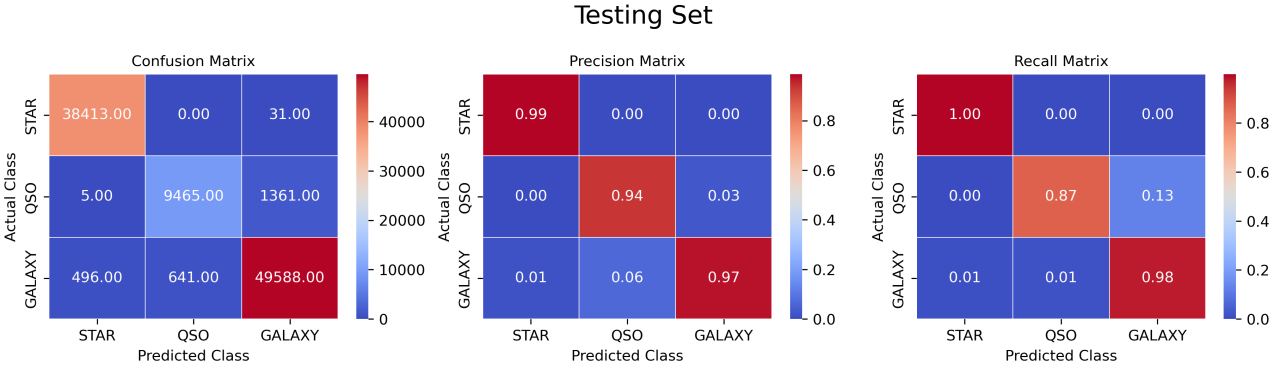
```
rfc = RandomForestClassifier(random_state=random_seed)
rfc.fit(X_train,y_train)
```

After training the model with default hyperparameters, accuracy score of 0.97 was obtained both on testing and validation datasets. Figure 4.4 shows the training, testing and validation confusion

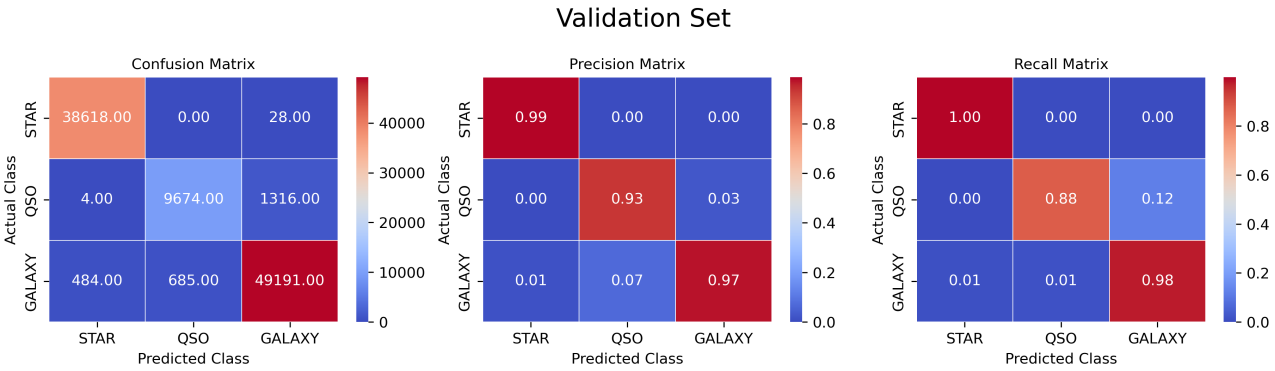
matrices with precision and recall scores. From the figure, we can see that the model is able to capture the patterns and accurately predict the class of object. Even the validation dataset has a good average recall of 0.95. However, due to such high scores one can suspect that overfitting might be taking place. To address this we can look at other evaulation metrics like precision and f1 score to determine if overfitting is the case or not.



(a) Training Set Confusion Matrix, Precision and Recall



(b) Testing Set Confusion Matrix, Precision and Recall



(c) Validation Set Confusion Matrix, Precision and Recall

Figure 4.4: Baseline performance of Random Forest Classifier

After establishing the baseline for Random Forest Classifier we can apply genetic algorithm to tune the hyperparameters. From subsection 2.3.1 we can choose hyperparameters and create a hyperparameter space. In the code block Listing 4.1 this hyperparameter space is named *param_grid* and I chose 2 hyperparameters due to computational limits. The sklearn-genetic-opt library offers a useful module named sklearn_genetic.space that serves as a valuable tool for defining the hyperparameter

space in which the genetic algorithm operates. Researchers and practitioners can specify the types of values that hyperparameters can take on during the optimization procedure using this module.

The module provides three distinct categories for defining the hyperparameter space:

- **Continuous Space**: When dealing with continuous hyperparameters, the 'Continuous' space option comes into play. This functionality enables the generation of values that are continuous, either as integers or floating-point numbers. These values are generated from a specified distribution, allowing for a broad range of possibilities. When a hyperparameter such as learning rate requires a continuous range of values, this option becomes extremely valuable. With the ability to consider both integers and floats, a variety of scenarios can be accommodated.

- **Integer Space**: In scenarios where hyperparameters must exclusively be integers, the 'Integer' space option becomes valuable. This feature ensures that only whole numbers are taken into account when optimizing. The 'Integer' space is an ideal choice for hyperparameters that involve discrete steps, such as the number of layers in a neural network or the number of trees in a random forest.

- **Categorical Space**: Some hyperparameters present themselves in a categorical format, where the choices are expressed as strings or labels. The 'Categorical' space option addresses such scenarios by allowing the specification of a list of possible values. This approach is particularly useful when dealing with hyperparameters that correspond to specific configurations or options. For instance, a choice between different activation functions in a neural network or different kernel types in a support vector machine can be handled through the 'Categorical' space.

Using these distinct space options, the sklearn-genetic-opt library ensures that the genetic algorithm may be tailored to meet the needs of various hyperparameter types, enhancing its adaptability and effectiveness. As a result of defining the hyperparameter space in this manner, the optimization procedure is more efficient because the algorithm will be able to navigate the search space more effectively. Thus, it is more likely that optimal hyperparameter settings will be found, leading to enhanced performance of the model.

I have used StratifiedKFold for cross validation as our dataset has imbalanced classes, see Figure 4.1. The function *GASearchCV( )* from sklearn-genetic-opt library is used to apply genetic algorithm on the hyperparameter space. The functions take values which are genetic operators as mentioned in detail in subsection 2.4.1. These operators are selected based on official documentation and also keeping in mind the computational limitations, as larger values significantly slow down the search algorithm. The code used here takes about 20 minutes to train.

```
param_grid = {'min_weight_fraction_leaf': Continuous(0.01, 0.5,
   distribution='log-uniform',
    random_state=random_seed),
    'n_estimators': Integer(100, 300,random_state=random_seed)}


```

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=random_seed)

rfc_genopt = GASearchCV(estimator=rfc,
                                cv=cv,
                                scoring='accuracy',
                                population_size=10,
                                generations=5,
                                tournament_size=3,
                                elitism=True,
                                crossover_probability=0.5,
                                mutation_probability=0.1,
                                param_grid=param_grid,
                                criteria='max',
                                algorithm='eaMuPlusLambda',
                                n_jobs=-1,
                                verbose=True,
                                keep_top_k=3)
# TAKES ABOUT 15-20 MINUTES TO RUN
rfc_genopt.fit(X_train,y_train)
```

Listing 4.1: Applying Genetic algorithm on Random Forest Classifier

Here's an explanation of each parameters used to instantiate the *GASearchCV( )* object:

- **estimator**: The machine learning model for which we want to optimize the hyperparameters. In this case it is 'rfc' standing for random forest classifier.
- **cv**: Cross-validation strategy to be used for evaluating different hyperparameter configurations. In this case we have 5 fold cross validation using StratifiedKFold. Stratification takes case of class imbalance by maintaing the proportion of classes in K folds.
- **scoring**: The scoring metric used to evaluate the performance of different hyperparameter configurations. Common options are accuracy, precison and recall.
- **population_size**: The number of individuals (candidate hyperparameter configurations) in each generation of the genetic algorithm.
- **generations**: The number of generations or iterations the genetic algorithm will run. Depending on available computing resources this number can be chosen by the practitioner. More the generations, better the search towards most optimum values.
- **tournament_size**: The size of tournament selection process, where individuals compete to be selected for crossover and mutation.
- **elitism**: A boolean flag indicating whether elitism is enabled. Elitism preserves the best individuals from each generation to ensure they are passed to the next generation unchanged. This

is analogous to the Darwin's rule of evolution - "The strong one survives". In this case the "strong" one is the one with maximum scoring metric.

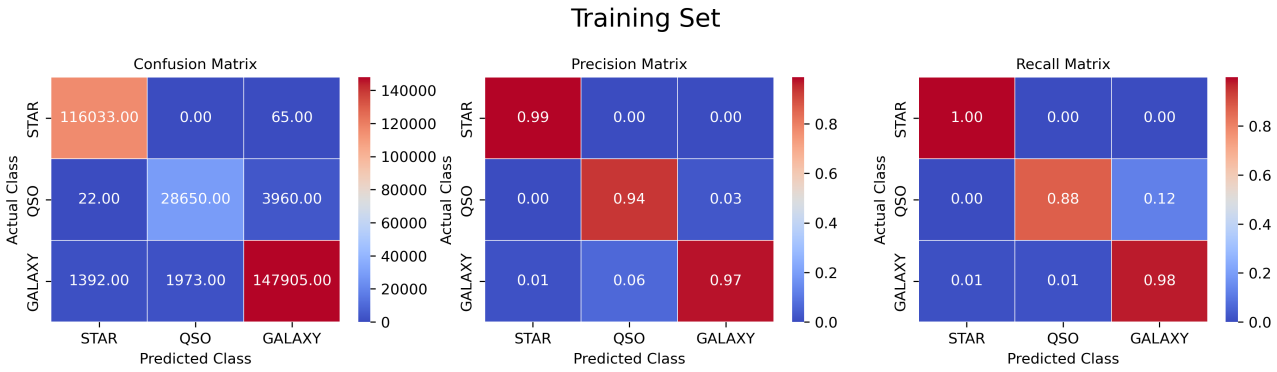
- **crossover_probability**: The probability of performing crossover(recombination) during genetic operations.
- **mutation_probability**: The probability of performing mutation on individuals during genetic operation. Higher probability leads to wider search space exploration, but too high of a value will lead to the algorithm not finding the most optimum solution.
- **param_grid**: The search space defined by the user with a range of values.
- **criteria**: The optimization criterion used to determine the best hyperparameter configuration. Options include 'max' (maximize the scoring metric) or 'min' (minimize the scoring metric).
- **algorithm**: The genetic algorithm variant to be used. Options include 'eaSimple', 'eaMuPlusLambda', 'eaMuCommaLambda', etc. 'eaMuCommaLambda' combines a mix of new individuals (lambda individuals) and the best individuals from the previous generation (mu individuals). However, in this case, the newly generated individuals completely replace the previous population. This can lead to more exploration of the solution space but may discard potentially good solutions from the previous generation.
- **n_jobs**: The number of CPU cores used for parallel computation (-1 indicates using all available cores).
- **verbose**: If True, it prints progress updates during optimization.
- **keep_top_k**: The number of top individuals from each generation that are preserved for the next generation.

Here's the verbose printed while the Genetic algorithm is being used to search the hyperspace:

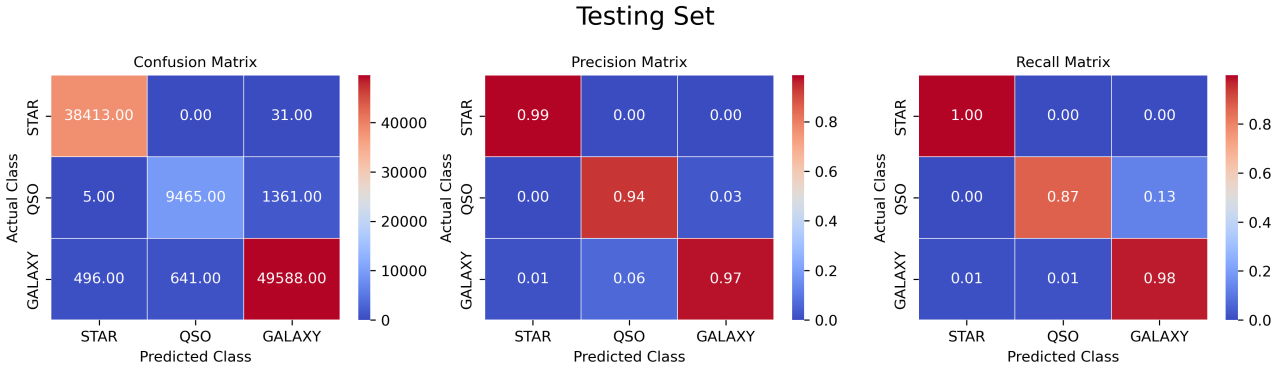
Table 4.2: Genetic Algorithm verbose for Random Forest Classifier

gen	nevals	fitness	fitness_std	fitness_max	fitness_min
0	10	0.947509	0.0487177	0.97494	0.823433
1	12	0.972395	0.00216501	0.97494	0.968843
2	9	0.973929	0.00182915	0.97494	0.9692
3	13	0.975158	0.000653	0.977117	0.97494
4	13	0.975744	0.000683351	0.977117	0.97494
5	14	0.976382	0.00048117	0.977117	0.976067

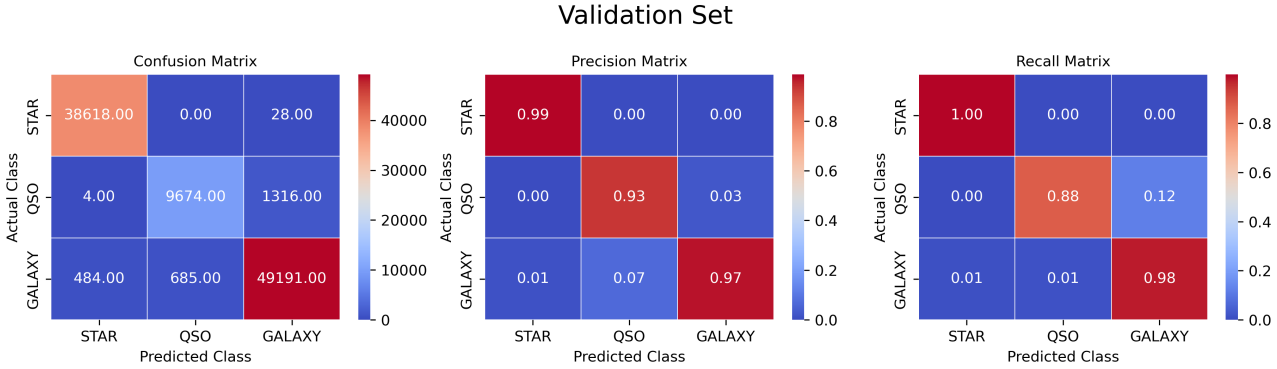
Figure 4.5 displays the performance of Random Forest model after genetic optimisation.



(a) Training Set Confusion Matrix, Precision and Recall



(b) Testing Set Confusion Matrix, Precision and Recall



(c) Validation Set Confusion Matrix, Precision and Recall

Figure 4.5: Performance of Random Forest Classifier after Genetic optimisation

Table 4.3: Top 3 best Hyperparameters for Random Forest Classifier

	1	2	3
min_weight_fraction_leaf	0.014455	0.014455	0.014455
n_estimators	255.000000	257.000000	125.000000

The default value of n_estimators is 100 (this was recently changed to 100 from 10) and min_weight_fraction_leaf is 0. But after using GA we are still getting very similar results if we round off to two decimal places. This mean either the change in values has no effect or our dataset is such that the model is able to learn or recognize the pattern very well. The later is more likely due to consistently high precision, recall and F1 scores both on testing and validation datasets.

Table 4.4 compares the metric values before (using default hyperparameters) and after (after genetic optimisation). We can see that there is hardly any change, the models are able to classify objects with very high precision and accuracy.

Table 4.4: Before and After comparison of random forest classifier metrics of Validation set

	Precision		Recall		F1-Score		Accuracy	
	Before	After	Before	After	Before	After	Before	After
GALAXY	0.97	0.97	0.98	0.97	0.98	0.98		
QSO	0.93	0.83	0.88	0.88	0.91	0.91	0.97	0.97
STAR	0.99	0.99	1.00	1.00	0.99	0.99		

4.2.2 Gradient Boost Classifier

We will follow the same framework as Random forest classifier, starting with establishing baseline performance of the model.

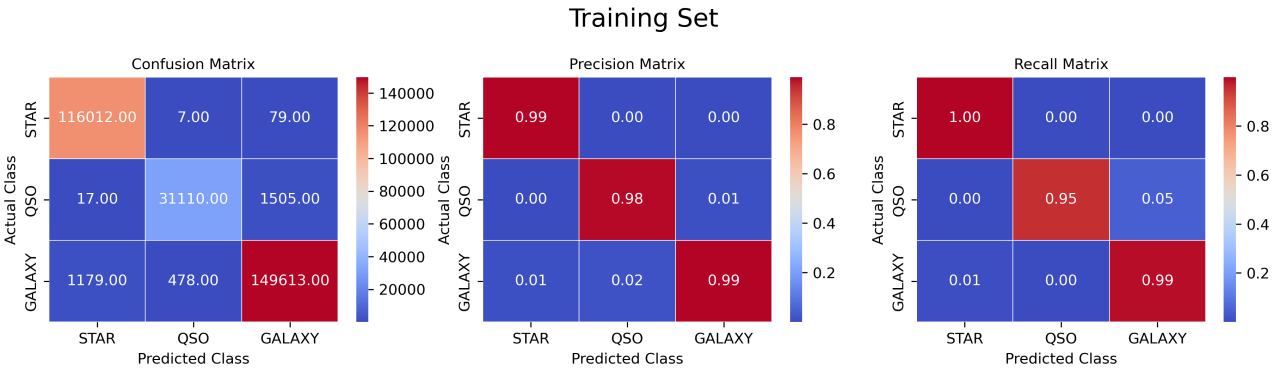
```
    gbc = GradientBoostingClassifier(random_state=random_seed)
    # TAKES 5 MINUTES TO RUN
    gbc.fit(X_train, y_train)
```

After training the model with default hyperparameters, on both testing and validation datasets we get accuracy score of ≈ 0.98.

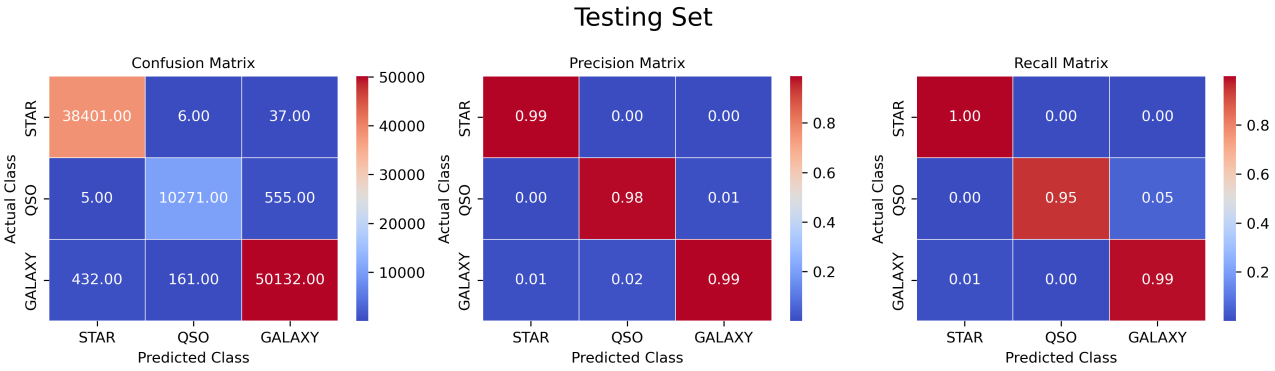
Figure 4.6 shows the confusion matrices with precision and recall scores for training, testing and validation scores. Even in this classifier we see good performance of model even on validation dataset.

From subsection 2.3.2 we can choose hyperparameters and create a hyperparameter space. Work by Clarke et al. (2020) gives us an idea of important hyperparameters which influence the performance of machine learning models. Based on that and consider limited computational resources, I selected 2 hyperparameters (learning rate and n_estimaters) to form my hyperparameter space. Even though learning rate is a float number, I have selected this hyperparameter as a categorical type consisting of 4 options, see Listing 4.2

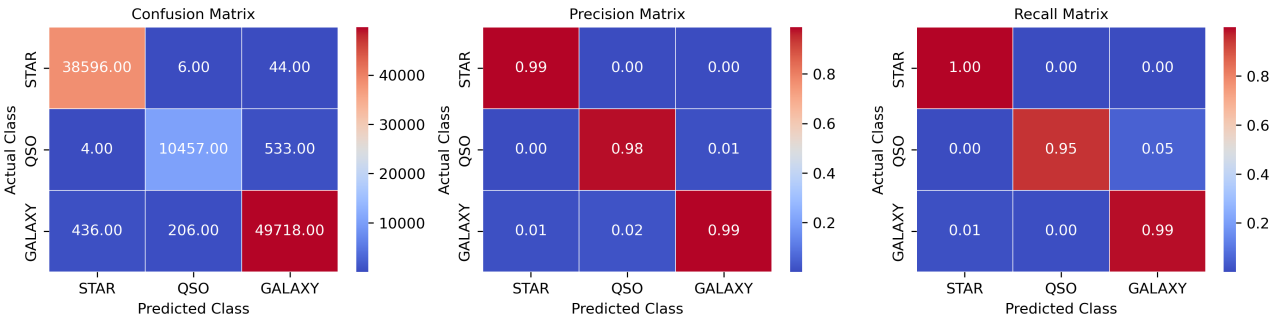
```
gbc = GradientBoostingClassifier(random_state=random_seed)

param_grid = {
    'learning_rate': Categorical([0.0001, 0.001, 0.01, 0.1],random_state=
   random_seed),
    'n_estimators': Integer(50, 200, distribution='uniform',random_state=
   random_seed)
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=random_seed)

gbc_genopt = GASearchCV(estimator=gbc,
```



(a) Training Set Confusion Matrix, Precision and Recall



(b) Testing Set Confusion Matrix, Precision and Recall



(c) Validation Set Confusion Matrix, Precision and Recall

Figure 4.6: Baseline performance of Gradient Boosting Classifier

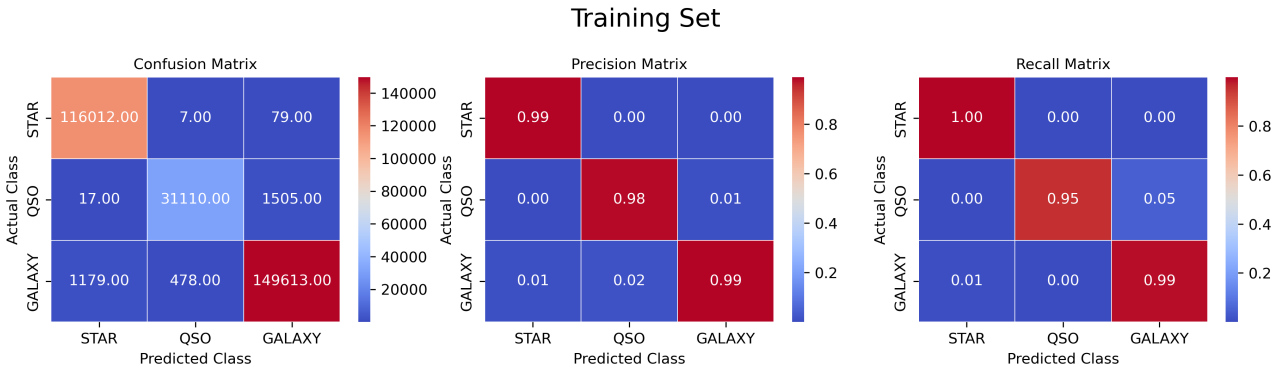
```
                                cv=cv,
                                scoring='accuracy',
                                population_size=5,
                                generations=5,
                                tournament_size=2,
                                elitism=True,
                                crossover_probability=0.5,
                                mutation_probability=0.1,
                                param_grid=param_grid,
                                criteria='max',
                                algorithm='eaMuPlusLambda',
                                n_jobs=-1,
                                verbose=True,
                                keep_top_k=3)
# TAKES ABOUT 240 MINUTES TO RUN
gbc_genopt.fit(X_train,y_train)
```

Listing 4.2: Applying genetic algorithm to Gradient Boosting Classifer

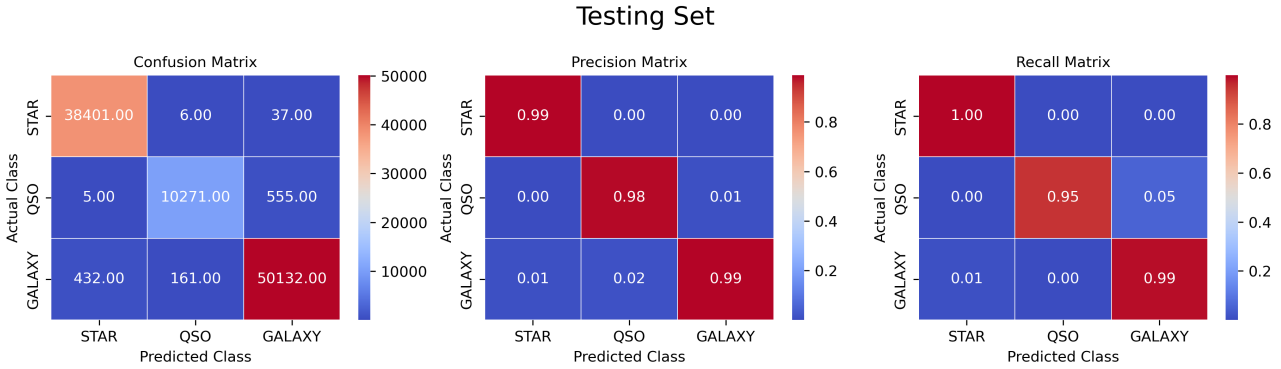
Table 4.5: Genetic Algorithm verbose for Gradient Boosting Classifier

gen	nevals	fitness	fitness_std	fitness_max	fitness_min
0	5	0.850162	0.178689	0.98788	0.504233
1	5	0.967104	0.0403195	0.98788	0.886483
2	6	0.987343	0.000877902	0.98788	0.985597
3	4	0.98777	0.000111355	0.98793	0.98768
4	5	0.98776	9.79796e-05	0.98788	0.98768
5	7	0.98784	8e-05	0.98788	0.98768

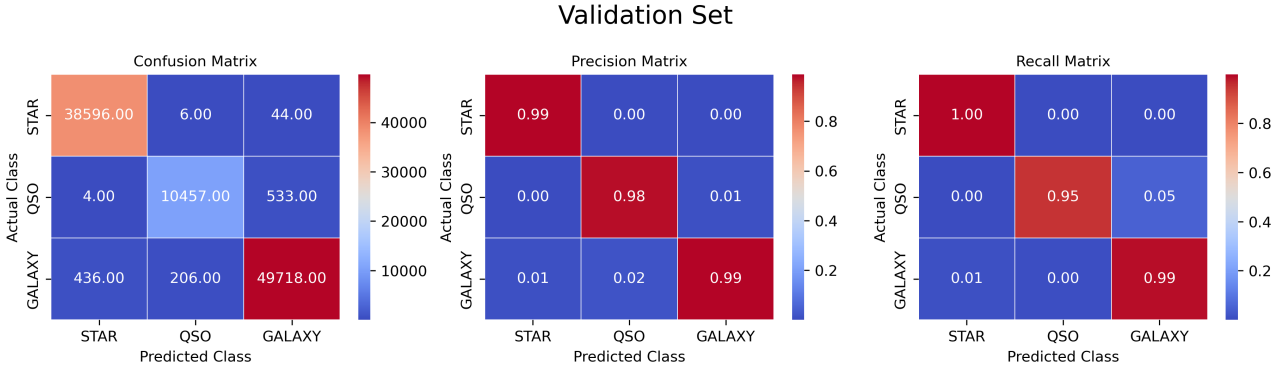
From the verbose we can observe that the fitness (accuracy) approaches a similar value obtained from using default hyperparameters as generations increase.



(a) Training Set Confusion Matrix, Precision and Recall



(b) Testing Set Confusion Matrix, Precision and Recall



(c) Validation Set Confusion Matrix, Precision and Recall

Figure 4.7: Performance of Gradient Boosting Classifier after Genetic optimisation

Table 4.6: Top 3 best Hyperparameters for Gradient Boosting Classifier

	1	2	3
learning_rate	0.1	0.1	0.1
n_estimators	179.0	166.0	155.0

The default value of n_estimators is 100 and of learning_rate is 0.1 which is consistent with what genetic optimisation yeilds. It is possible that the default hyperparameters are optimum for the dataset and it is able to learn the underlying patterns leading to such high metric scores in classification.

Table 4.7: Before and After comparison of random forest classifier metrics of Validation set

	Precision		Recall		F1-Score		Accuracy	
	Before	After	Before	After	Before	After	Before	After
GALAXY	0.99	0.99	0.99	0.99	0.99	0.99		
QSO	0.98	0.98	0.95	0.95	0.97	0.97	0.99	0.99
STAR	0.99	0.99	1.00	1.00	0.99	0.99		

Such high accuracy scores definitely raise suspicions of overfitting. To address this issue we considered Testing and validation set metrics and even they have very high values. In such a scenario it is possible that the data is such that the patterns are learned by the models resulting in highly accurate classification even on unseen dataset. In case of Gradient Boosting Classifier the precision, recall and F1 scores are exactly the same. The code with default hyperparameters takes about 2% of the time it takes for genetic algorithm to search through the hyperparameter space. This indicates that the default values themselves are optimised values, especially for datasets with apparent patterns.

4.2.3 Logistic Regression

As a means of verifying the results I observed in the previous two models, I chose to use Logistic Regression. The logistic regression model uses regularisation and penalties to mitigate overfitting if it occurs in the previous two models. I was unsuccessful in my attempt to optimize hyperparameters of the Logistic Regression using genetic algorithms. Due to unknown reasons, sklean-genetic-opt was not working properly with logistic regression. There may be a compatibility issue between the penalties and the kind of solver used. The hyperparameter search space could not be explored due to the fact that not all solvers support all penalties. This means that problems of hyperparameter compatibility cannot be considered when exploring the hyperparameter search space.

I decided to proceed with default parameters as they have lead to high accuracy scores on previous models.

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=random_seed)
clf = LogisticRegressionCV(Cs=10, cv=cv, penalty='l2',solver='saga',
   n_jobs=-1,random_state=random_seed)
clf.fit(X=X_train, y=y_train)
```

Listing 4.3: Logistic Regression with default hyperparameters

Table 4.8: Classification Report for Logistic regression on validation set

	Precision	Recall	F1-Score
GALAXY	0.99	0.98	0.98
QSO	0.98	0.94	0.96
STAR	0.97	1.00	0.99

Upon reviewing the validation set classification report, it is evident that we still have high precision, recall, and F1 scores, suggesting that this is likely not the result of overfitting, since all models are giving high scores. This may be due to apparent patterns in the dataset, and such a situation may be unique to this type of dataset.

4.3 Limitations

A prominent constraint that emerged during the study was the imposition of computational limitations. Genetic algorithms necessitate extensive parallel processing to efficiently explore the hyperparameter space. Despite this challenge, genetic algorithms are known for their capability to navigate the hyperparameter landscape and converge towards optimal values. This contrasted with alternative techniques such as random or grid search, which lack the assurance of identifying optimal solutions.

In addition, it is difficult to determine whether hyperparameters are compatible with each other since the search space is determined by the user. The ability to identify which hyperparameters do not work with others will be helpful, but it will also hinder the process of finding the most optimum value.

Chapter 5

Conclusions

The primary objective of this thesis was to determine whether genetic algorithms could be utilized to tune hyperparameter values. In the event that it's feasible, what improvement can be achieved in the models given the computational cost. Research began by reviewing literature and finding similar works published elsewhere. Based on these papers, three machine learning models are selected keeping in mind the computational constraint and the time constraint. Observations conducted by large-scale astronomical surveys record data at an astronomical scale. This data consists of all sorts of astronomical objects and research groups all over the world require data as per their requirements. As a result, the classification of astronomical objects and the creation of detailed catalogues are among the most important and challenging tasks of such observatories. Every time the data is released, millions of new objects are required to be classified, and machine learning models are well suited for this task because they can be integrated with the data pipeline. The level of automation entails the use of reliable models that are tuned to the specific type of data being handled.

5.1 Model performance

The baseline performance, i.e., using default hyperparameters of the models of choice, led to very high accuracy scores. This raised the suspicion that the model was overfitting the data and would not be able to generalize, thus resulting in poor classification performance on new data. The study examined concepts such as precision, recall, F1-score, and log loss, and demonstrated the importance of these metrics in quantifying the performance of classification models. The issue of class imbalance emerged as a significant concern, echoing a common real-world scenario in which certain classes have disproportionately fewer instances. In order to mitigate these biases and inaccuracies, stratified sampling was introduced.

In an intriguing discovery, it was found that models using default hyperparameters and genetic algorithms produced similar results. There are several aspects to consider in light of this discovery. In our context, this could very well demonstrate how well-suited the default hyperparameters are to the classification problem by demonstrating their robustness to the particular domain of our dataset. Alternatively, it may indicate that hyperparameter selection and genetic algorithm configurations had no effect on the model's performance. It emphasizes the importance of selecting hyperparameters

carefully and fine-tuning them according to domain knowledge.

5.2 Ethical considerations

Despite the fact that my thesis was based on a publicly available astronomical dataset, it is still important to address ethical considerations. While I did not encounter any specific ethical concerns during the course of my research, it is essential to note and discuss the potential ethical implications that may arise in similar studies. The use of a publicly available astronomical dataset may alleviate some ethical concerns, it is still essential to address potential ethical considerations related to data privacy, biases, recognition of contributors, and academic integrity. This thesis utilized a dataset in which no information pertaining to any person, organization, or entity could potentially violate their privacy. Observatories and funding agencies are deeming the SDSS data to be free for public use provided that credit is given where credit is due.

5.3 Closing remarks

This research produced insightful findings and it paves the way for more investigation in the future. It is necessary to investigate increasingly complex hyperparameter spaces and investigate different optimization strategies. With advances in parallel computing technologies, searching complex hyperparameter spaces will become more and more efficient. Future studies are also anticipated to investigate the use of genetic algorithms for a wider variety of optimisation problems than only categorization problems. This was an exciting journey to study the intersection between machine learning, genetics, and astronomical exploration.

Bibliography

Almeida, A. and et. al (2023), 'The Eighteenth Data Release of the Sloan Digital Sky Surveys: Targeting and First Spectra from SDSS-V', *arXiv e-prints* p. arXiv:2301.07688.
URL: https://doi.org/10.48550/arXiv.2301.07688

Barve, P. (2023), 'GitHub - iamstarstuff/MScDataScienceThesis', https://github.com/iamstarstuff/MScDataScienceThesis. [Accessed 14-08-2023].

Becker, I., Pichara, K., Catelan, M., Protopapas, P., Aguirre, C. and Nikzat, F. (2020), 'Scalable end-to-end recurrent neural network for variable star classification', *Monthly Notices of the Royal Astronomical Society* **493**(2), 2981–2995.
URL: https://doi.org/10.1093/mnras/staa350

Breiman, L. (2001), 'Random forests', *Machine Learning* **45**(1), 5–32.
URL: https://doi.org/10.1023/A:1010933404324

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B. and Varoquaux, G. (2013), API design for machine learning software: experiences from the scikit-learn project, *in* 'ECML PKDD Workshop: Languages for Data Mining and Machine Learning', pp. 108–122.
URL: https://doi.org/10.48550/arXiv.1309.0238

Clarke, A. O., Scaife, A. M. M., Greenhalgh, R. and Griguta, V. (2020), 'Identifying galaxies, quasars, and stars with machine learning: A new catalogue of classifications for 111 million sdss sources without spectra', *Astronomy & Astrophysics* **639**, A84.
URL: https://doi.org/10.1051/0004-6361/201936770

Cortes, C. and Vapnik, V. (1995), 'Support-vector networks', *Machine Learning* **20**, 273–297.

Cui, H. and Bai, J. (2019), 'A new hyperparameters optimization method for convolutional neural networks', *Pattern Recognition Letters* **125**, 828–834.

Deng, H., Zhou, Y., Wang, L. and Zhang, C. (2021), 'Ensemble learning for the early prediction of neonatal jaundice with genetic features', *BMC Medical Informatics and Decision Making* **21**.
URL: https://doi.org/10.1186/s12911-021-01701-9

Dodson, M. (1976), 'Darwin's law of natural selection and thom's theory of catastrophes', *Mathematical Biosciences* **28**(3), 243–274.

fast.ai contributors (2023), 'Overfitting and Underfitting — fastaireference.com', https://www.fastaireference.com/overfitting. [Accessed 10-08-2023].

Friedman, J. H. (2001), 'Greedy function approximation: A gradient boosting machine', *The Annals of Statistics* **29**(5), 1189–1232.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. and Oliphant, T. E. (2020), 'Array programming with NumPy', *Nature* **585**(7825), 357–362.
URL: *https://doi.org/10.1038/s41586-020-2649-2*

Hastie, T., Tibshirani, R. and Friedman, J. (2009*a*), Chapter 10: Boosting and additive trees, *in* 'The elements of statistical learning: data mining, inference and prediction', 2 edn, Springer, pp. 337–387.

Hastie, T., Tibshirani, R. and Friedman, J. (2009*b*), Chapter 7: Model assessment and selection, *in* 'The elements of statistical learning: data mining, inference and prediction', 2 edn, Springer, pp. 219–259.

Hawkins, D. M. (2004), 'The problem of overfitting', *Journal of Chemical Information and Computer Sciences* **44**(1), 1–12.
URL: *https://doi.org/10.1021/ci0342472*

Hennings, V. (2013), 'Victor Henning's brief guide to Mendeley – elsevier.com', https://www.elsevier.com/connect/archive/victor-hennings-brief-guide-to-mendeley. [Accessed 10-08-2023].

Ho, T. K. (1995), Random decision forests, *in* 'Proceedings of 3rd International Conference on Document Analysis and Recognition', Vol. 1, pp. 278–282 vol.1.

Holland, J. H. (1992), 'Genetic algorithms', *Scientific American* **267**(1), 66–73.

Hosmer, D. W. and Lemeshow, S. (2000), *Applied logistic regression*, 2 edn, John Wiley and Sons.

Hubble, E. P. (1926), 'Extragalactic nebulae.', *ApJ* **64**, 321–369.

Hunter, J. D. (2007), 'Matplotlib: A 2d graphics environment', *Computing in Science & Engineering* **9**(3), 90–95.

Jebari, K. (2013), 'Selection methods for genetic algorithms', *International Journal of Emerging Sciences* **3**, 333–344.

Jin, X., Zhang, Y., Zhang, J., Zhao, Y., Wu, X.-b. and Fan, D. (2019), 'Efficient selection of quasar candidates based on optical and infrared photometric data using machine learning', *Monthly Notices of the Royal Astronomical Society* **485**(4), 4539–4549.

Katoch, S., Chauhan, S. S. and Kumar, V. (2021), 'A review on genetic algorithm: past, present, and future', *Multimedia Tools and Applications* **80**, 8091–8126.

Khalifa, N. E., Hamed Taha, M., Hassanien, A. E. and Selim, I. (2018), 'Deep galaxy v2: Robust deep convolutional neural networks for galaxy morphology classifications', pp. 1–6.

Kim, E. J. and Brunner, R. J. (2016), 'Star-galaxy classification using deep convolutional neural networks'.

Liu, D. (2019), 'Mathematical modeling analysis of genetic algorithms under schema theorem', *Journal of Computational Methods in Sciences and Engineering* **9**, 131–137.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011), 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research* **12**, 2825–2830.
URL: *https://doi.org/10.48550/arXiv.1201.0490*

Powers, D. (2011), 'Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation', *International Journal of Machine Learning Technology* **2**, 37–63.
URL: *https://doi.org/10.48550/arXiv.2010.16061*

Rokach, L. (2010), 'Ensemble-based classifiers', *Artificial Intelligence Review* **33**(1), 1–39.
URL: *https://doi.org/10.1007/s10462-009-9124-7*

Saiffudin, M. (2022), 'Stellar Classification: A Machine Learning Approach –towardsdatascience.com', https://towardsdatascience.com/stellar-classification-a-machine-learning-approach-5e23eb5cadb1. [Accessed 12-08-2023].

Scikit-learn contributors (2023), 'Cross-validation', https://scikit-learn.org/stable/modules/cross_validation.html. [scikit-learn 1.3.0 documentation; Online; accessed 10 August 2023].

SDSS.org (2022), 'SQL Search - SkyserverSDSS – skyserver.sdss.org', https://skyserver.sdss.org/dr18/SearchTools/sql. [Accessed 10-08-2023].

Singhal, G. (2020), 'Ensemble methods in machine learning: Bagging versus boosting', https://www.pluralsight.com/guides/ensemble-methods:-bagging-versus-boosting. Accessed: 06 August 2023.

Sloan Digital Sky Survey | Alfred P. Sloan Foundation (n.d.), https://sloan.org/programs/research/sloan-digital-sky-survey. [Accessed 13-08-2023].

Tang, K., Man, K., Kwong, S. and He, Q. (1996), 'Genetic algorithms and their applications', *IEEE Signal Processing Magazine* **13**, 22–37.

Tharwat, A. (2021), 'Classification assessment methods', *Applied Computing and Informatics* **17**, 168–192.
URL: *https://doi.org/10.1016/j.aci.2018.08.003*

Wes McKinney (2010), Data Structures for Statistical Computing in Python, *in* Stéfan van der Walt and Jarrod Millman, eds, 'Proceedings of the 9th Python in Science Conference', pp. 56 – 61.
URL: *http://doi.org/10.25080/Majora-92bf1922-00a*

Wierzbiński, M., Pławiak, P., Hammad, M. and Acharya, U. R. (2021), 'Development of accurate classification of heavenly bodies using novel machine learning techniques', *Soft Computing* **25**, 7213–7228.

Young, S. R., Rose, D. C., Karnowski, T. P., Lim, S.-H. and Patton, R. M. (2015), 'Optimizing deep learning hyper-parameters through an evolutionary algorithm', *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments* .

Zhang, Y., Zhao, Y., Zheng, H. and bing Wu, X. (2012), 'Classification of quasars and stars by supervised and unsupervised methods', *Proceedings of the International Astronomical Union* **8**, 333–334.