Predicting Redshifts & Morphology of Galaxies with Flux Magnitudes.

Application of Regression & Classification with Comparative Analysis between Decision Tree & Random Forest as Ensemble Classifier.

Biju Ale
aleb@coventry.ac.uk

Faculty of Engineering, Environment and Computing
School of Computing, Electronics and Mathematics
Coventry University

October 19, 2019

Dissertation for BSc (Honours) in Computing

Abstract

Light from distant galaxies is a goldmine of information to determine attributes such as its redshifts and even its morphology. However, measuring the individual shift in wavelengths based on the localized spectral fingerprint of an associated element is not feasible when hundreds of billions of galaxies populate space. And in other cases, spectroscopic observations are simply unavailable for many galaxies. In this paper, the Decision Tree regression classifier is trained with the color indices at 5 bands with the corresponding photometric redshifts. The performance of the learning model is assessed with the median residual method. 10-fold cross-validation is used to improve its performance. Quasars also populated the dataset with median residual after regression at ≈ 0.074 and for galaxies at ≈ 0.016. The spectrum flux is also used in predicting the morphology of galaxies. With the decision tree, 80% accuracy compared to true values of morphologies, was achieved. Ensemble Learning using the random forest improved the accuracy by 6-7%.

Keywords: Machine Learning, Astronomy, Redshift, Morphology, Decision Tree, Random Forest, Galaxy, Ensemble Method, Sloan Digital Sky Survey (SDSS), Galaxy Zoo, Hubble Sequence, Data-driven Astronomy

To my mother, S. Ale

Contents

List of Figures

List of Tables

1 Introduction

> "Two things fill the mind with ever new and increasing admiration and awe, the more often and steadily we reflect upon them: the starry heavens above me and the moral law within me."
>
> Immanuel Kant (1724-1804)
> *Critique of Practical Reason, 1788*

1.1 Background

The notion of colour sends a philosopher pondering about its ontology and the relation with the mind, while the impressionists akin to Monet, are busy evoking awe in the play with the pigments. And there are the astronomers who have precisely defined colours in terms of the electromagnetic spectrum[1] of light. Alex Filippenko, the professor of astronomy at UC Berkeley put it aptly; light is the supreme informant, a goldmine for astronomers. Its colours are the embedded knowledge of the celestial bodies (Filippenko 2007: 96).

Astronomers have been training their telescopes to collect light from galactic objects and the quality and quantity of data have improved with the improvement of their telescopes from Galileo's refractive lens to the modern adaptive optics and onward to the more advanced charged-couple detectors. The last decade has seen a proliferation of computing power, combined with modern astronomy's advanced high-throughput telescopes, the summation of data generated is humongous (Ivezić, Connolly, VanderPlas, et al. 2014: 43). Thence emerges an inevitable problem in analysing these vast heaps of data and then mining out the embedded knowledge to facilitate further predictions.

In this study, I use the publicly released dataset on distant galaxies, administered by and collected from the *Sloan Digital Sky Survey* (Stoughton, Lupton, Bernardi, et al. 2002) including the public citizen science project, *Galaxy Zoo* (Lintott, Schawinski, Slosar, et al. 2008; Raddick, Bracey, Gay, et al. 2010; Lintott, Schawinski, Bamford, et al. 2011; Raddick, Bracey, Gay, et al. 2013). I look particularly at the flux magnitudes of galaxies and take in as features the magnitudes across 5 colours or bands in the electromagnetic spectrum. Using the co-relation between flux magnitudes and redshifts, similarly flux magnitudes with the morphology of the galaxy, I employ machine learning models to predict the target features, namely redshifts and morphology.

1.2 Aim & Objectives

The primary aim of my dissertation is to discover whether ensemble classifiers improve the prediction accuracy of galaxy redshifts and morphology, based on their apparent flux magnitudes. And if so, by what factor?

In the process, I strive to garner skills in applied machine learning and also insights in astronomy. Other learnings include the method of conducting empirical research while developing technical soundness and an incisive mind for problem-solving. Moreover, I regard this dissertation as an encouragement for students to engage in multi-disciplinary approach to problem resolution.

1.2.1 Technical Objectives.

1. Conduct Literature Review
   (a) Review astronomy research on galaxy spectroscopy.
   (b) Review Machine Learning applications in astronomy.
2. Perform Regression to Predict Redshifts
   (a) Prepare SDSS dataset.
   (b) Perform Hold-out Validation.
   (c) Model's performance.
   (d) Overfitting assessment.
   (e) Optimum Tree-depth.
   (f) Perform K-Fold Cross Validation.
3. Perform Classification to predict Galaxy Morphology.
   (a) Prepare Galaxy Zoo dataset.
   (b) Engineer & Extract Features.
   (c) Perform hold-out validation.
   (d) Assess decision tree's performance.
   (e) Perform Hold-out Validation.
   (f) Perform Ensemble learning with Random Forest.
   (g) Assess Random Forest's Performance.
   (h) Perform Compare Performance of DT & RF.

1.2.2 Personal Objectives

1. Understand the nature of learning.
2. Be able to identify ML solvable problems
3. Garner Applied ML Skills
4. Foster an incisive mind
5. Garner applied ML skills
6. Build a scholarly Portfolio
7. Assimilate cosmological studies

1.2.3 Academic Objectives

1. Grok Research Methodology
2. Grok Data-driven Astronomy
3. Learn & Apply scientific computing with Python.
4. Cognize machine learning techniques.
5. Synthesize Knowledge.
6. Fulfill the Requirement for the Honors degree.
7. Qualify as a Graduate.

[1] The dispersion prism breaking light into its respective colours are famously illustrated in pop-culture. For instance, in an album cover of the English rock band Pink-Floyd. This phenomenon was demonstrated and explicated by Newton 300 years ago.

1.3 Overview

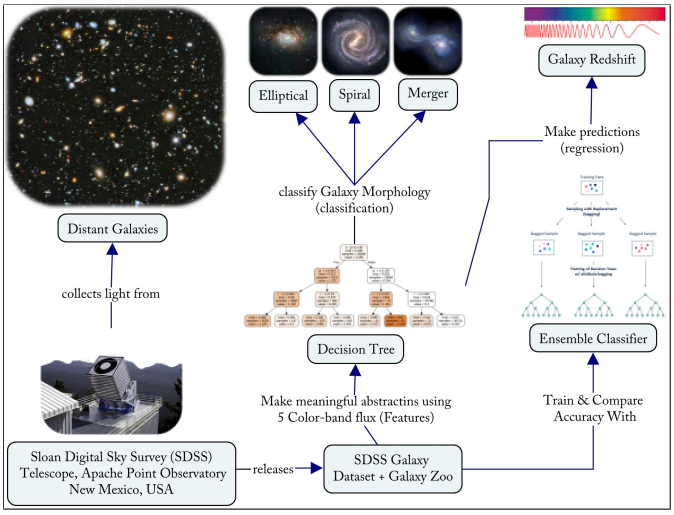


Figure 1: Rich Picture depicting the overview of the dissertation project. Image Credit/Reference: Distant Galaxies (NASA and Space Telescope Science Institute 2004), Elliptical (NASA and ESA/Hubble 2016), Spiral (NASA/JPL 2019), Merger (Evan Gough and National Astronomical Observatory of Japan (NAOJ) 2019), Sloan Telescope (IEEE Computer Society Digital Liibrary 1999)

One of the frustrations of an observational science like astronomy is we can't set up experiments in the lab. We can't control the unknown parameters to test hypotheses or use physical equipment like rulers to measure distances. Instead we have to cleverly construct observations to collect data that will allow us to answer our questions.

It is relatively straightforward for a person with the help of a computer to measure a redshift from a galaxy with an observed spectrum and hence work out how far away the galaxy is. But many galaxies have not been observed spectroscopically, we only have images. In addition, the sheer number of galaxies in these surveys makes this task impractical to do by hand. In the first part of my research, I shall task machine learning to calculate the redshift of galaxies from their measured colours .

In this study, I shall employ decision trees to calculate the redshift of galaxies from the Sloan Digital Sky Survey. Let's use this example to work through what the process of supervised machine learning looks like. Getting the methodology right is critical and we'll discuss some of the pitfalls in later sections. I start with a sample of galaxies for which the answer is already known. Such a sample is called the gold standard. These are galaxies with measured spectroscopic redshifts. Typically quite a large sample is needed because the learner needs to be able to see enough instances of each type of object and its properties to correctly model and then predict the correct results. In addition to the regression of redshifts, I also task the decision tree to classify galaxies based on their morphologies. The same features of flux across the colour bands are used.

1.4 Feature Extraction

The next step is to extract features that represent the input data in some way. Sometimes it is obvious how to do this because the manual classification scheme we use is designed by specific features. In this study, colours measured by comparing the magnitudes from five different Sloan filters. U, G, R, I and Z are used. Each filter measures the light from the galaxy in a particular wavelength range. To calculate colours, astronomers subtract the magnitudes measured in neighbouring filters. For example, U minus G or I minus Z.

1.5 Algorithm Selection

Decision tree is selected as the regresison classifier. Other algorithms like neural networks, random forests and naive Bayes classifiers can also be used. For some tasks, Naive Bayes classifiers are typically very fast but don't have great accuracy. Whereas random forests classify have high accuracy but can be much slower to run. One of the reasons decision trees are great is that the model they produce is quite intuitive for humans to understand. In fact, the decision tree was involved in the kinds of schemes scientists have been using for generations. For example, in developing taxonomies in biology.

1.6 Training & Accuracy

The process of training basically consists of building a model between the inputs and the result. The nature of the model depends on the classification algorithm being used, the big question when at the onset of training a classifier is, how accurate is it? It is worth thinking about how it applies in light of humans. In the early days of classification, most objects would only have been classified once. Classifiers like Annie Cannon were extremely expert and very careful. So their classifications were probably pretty reliable. But there's no doubt that even the most diligent human classifiers can make mistakes. An obvious way of evaluating the reliability of a human classifier would be to get another expert to classify the same set of data and compare their results. This would give a measure of how reliable their classifications are. If a classifier makes a mistake, an object of category A could be accidentally classified as category B. When all objects must be classified, each mistake introduces two classification errors, an extra B, a false positive and a missing A, a false negative. Machine learning researchers typically quantify the reliability of a classifier using two measures, precision and recall, and we'll explore these in the activities.

1.7 Cross-Validatioin

So how do we know our classifier will be accurate on unknown data if it's just trained on one data set? To answer this, I have used n-fold cross-validation. You split your sample of galaxies with known redshifts into ten sets or folds, which are usually chosen randomly. You then train your classifier on the galaxies in nine of these sets and test it on the galaxies in the tenth set and record the precision and recall. You repeat this process for each fold and calculate the mean and the standard deviation, which is how you get the name 10-fold Cross-Validation. You've now developed a system that learns the classificational regression from data and can evaluate how reliable it is. You can now run it on some unknown galaxies, determine their redshifts and estimate to make their errors.

2 Ethical & Professional Consideration

> "The integrity of the upright guides them, but the unfaithful are destroyed by their duplicity."
>
> Proverbs 11:3

The domain of Artificial Intelligence and Machine Learning has seen ethical repercussions when left unchecked. From the conspicuous case of a pedestrian fatality in 2018 in Arizona involving an autonomous vehicle (Wakabayashi 2018), to a more subtle case of inheritance of systemic bias into a machine learning algorithm (Shaw 2018), it is evident that the fragility of machines can take its toll on humanity. And this is where human's exceptional realization of objective morality needs to apply normative guidelines of ethics in building intelligent systems.

My research is largely devoid of human subjects and does not cater to service provision akin to commercial value gains or business optimisation. Rather it is purely a scientific undertaking of empirical studies involving cosmic objects, namely galaxies, their properties, and their data manipulation. In this regard, much of the ethical issues such as privacy concerns and data protection issues are more critical for example in health care systems than academic studies as this. However, this postulate should not be assumed to suggest indifference to ethical or professional considerations. It is just that they vary depending on the nature of the project.

In my research and design, I have followed the **8 principles** of responsible machine learning activity developed by the British think-tank, 'The Institute for Ethical AI & Machine Learning' focused on developing practical frameworks for ethical intelligent computing (2019).

1. **Human Augmentation**: I commit to assess the impact of failed predictions and introduce human-in-the-loop review.
2. **Bias Evaluation**: I commit to constantly monitor and record biases in my machine learning application's predictions during training and production.
3. **Explain ability By Justification**: I commit to promote transparency in the working mechanisms of my machine learning system with the aid of any tools and processes I can avail.
4. **Reproducible Operations**: I commit to document and explain with illustrations how my system was built from the initiation to the deployment.
5. **Displacement Strategy**: I commit to partner with domain experts and relevant stakeholders to explain my system and elicit approval from legal and professional bodies if its deployment involves a change in the human resource structure in the area of deployment.
6. **Practical Accuracy**: I commit to ensuring my model's predictive accuracy by leveraging the optimum number of approaches, strategies, and tools in my reach while I explain accuracy and implication of predictions with visuals such as confusion matrices or illustrations.
7. **Trust By Privacy**: I commit to being transparent with the dataset and their source I employ to train my models. I shall build trust with pertinent stakeholders by clearly informing them of the ways the data is being manipulated.
8. **Data Risk Awareness**: I commit to design, develop, and improve the processes and resources to ensure integrity, confidentiality, and authenticity of my data.

3 Literature Review

> "We are drowning in data, but starving for knowledge."
>
> John Naisbitt
> *Megatrends, 1982*

3.1 Data-driven Astronomy

Figure 2: Annie J. Cannon at Radcliffe College (Schlesinger Library 2016)

Humans are exceptional pattern matchers. Studies have shown that we are able to discern human faces within hundreds of milliseconds from first perceiving them (Caharel, Ramon, and Rossion 2013; Barragan-Jason, Cauchoix, and Barbeau 2015). Adapting this innate trait combined with the propensity to explain external reality, and with incisive reasoning; scientists have yielded groundbreaking discoveries. Consider the discovery of the laws of planetary motion by Johannes Kepler. The laws were the result of analysis of Tycho Brahe's comprehensive data on Mars' motion which he had catalogued diligently over two decades.

Likewise, Annie Jump Cannon (1863-1941) despite her hearing-impairment devised the stellar classification scheme also known as the 'Harvard Spectral Classification' (Wolbach Library of the Harvard-Smithsonian Center for Astrophysics 2019). With this scene, along with other human computers, she manually categorized a staggering quarter of a million stars.

As the efficiency of optics and computing technology catapults exponentially, the volume of data modern telescopes generates are dramatic. Within the last decade, surveys like the Sloan Digital Sky Survey and the Hubble Deep Field project have collected hundreds of terabytes of information (Ivezić, Connolly, VanderPlas, et al. 2014: 43). Criteria and heuristics are easily programmable while intuitions shown for example by Cannon is difficult to describe in a set of rules. As the rules grow with the complexity of the problem, it gets difficult to encapsulate the desired goal. Machine Learning addresses this issue by scaling learnability and applying the new-found knowledge to solve for unknown variables.

3.2 Machine Learning - A Primer

Computational Pattern Recognition The term 'Machine Learning' (ML) at its face value can seem oxymoronic to a layperson. How can insentient machines exhibit human attribute akin to learning? It cannot, in the way humans do. However, a more accurate definition would include the

potential for computer programs to detect pattern or trend in a given sequence of data. Therefore, machine learning all in all can be mapped to algorithmic *pattern recognition*.

Inductive inference Philosophers since antiquity have pondered upon and produced myriad frameworks for sound and valid inferences. From Aristotelian syllogism to the more extensive Fregean derivatives such as first-order predicate logic, the reasoning from induction has allowed to generalise out insightful functions from known data (Thagard 1990). Tom Mitchell a luminary in Machine Learning from Stanford, states that the inductive learning hypothesis generates such generalisations and is one of the fundamental assumptions in ML to facilitate predictions (1997: 23). This philosophical abstraction is realized by two major steps in ML. 1) Read a dataset (measurements with correlations across various physical phenomena). And 2) Build a statistical model that describes the dataset. The model obtained is assumed (reasonably so) to work for new data.

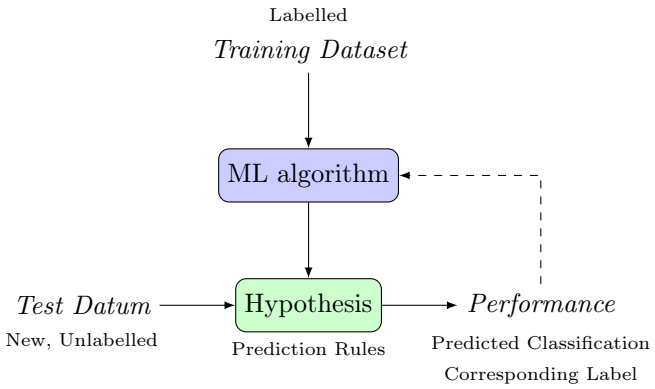


Figure 3: Machine Learning Process Abstraction

One of the earliest machine learning programs dates back to 1952 whence the AI pioneer, Carnegie Mellon professor Arthur L. Samuel (1901-1990) devised a remarkable computer game in the IBM's Poughkeepsie Laboratory (Stanford University 2019). Samuel's checker's playing program could learn from past experience to improve its moves (McCarthy and Feigenbaum 1990). His 1959 paper regarded seminal in the ML domain, set the groundwork for machine learning application and thus the term itself was coined for the first time (Samuel 1959). Samuel's definition of machine learning can be recapitulated as the domain of computing that enables programs to learn without the input of active and explicit instructions.

Modernizing Samuel's definition, the Carnegie Mellon computer scientist Tom Mitchell provides a more formal definition: 'A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E' (1997). Consider the example of a checkers playing machine learning program. This problem would be posed formally in Machine Learning terms as:

Experience	E	Experience of Checkers playing
Task	T	Playing Checkers
Performance	P	Probability of program's future victory

Table 1: Mitchell's formalisation of checker's playing ML program(1997: 6) .

3.3 Machine Learning's Relation with other domains

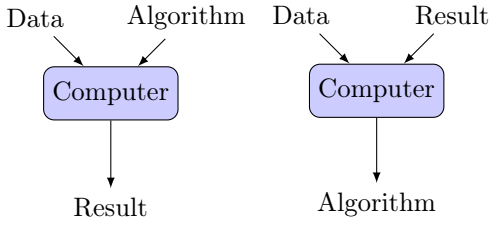


Figure 4: Traditional Programming vs Machine Learning

At the onset, it is crucial to differentiate traditional computing with machine learning. As depicted in the figure above, the results and raw materials generating them are conspicuously the inverted reflections of each other. However, programming is not replaced by machine learning. Rather they exist concurrently, and work complementing each other. The former solves the computational problem where the rules are known before-hand, and the later solves problems based on experience as heuristics programming is not possible for certain problems (Mitchell 1997).

Another important distinction is of that between AI and Machine Learning. The term Artificial Intelligence (AI) and Machine Learning are sometimes used interchangeably in a colloquial setting. Pedro Domingoes, professor of computer science and engineering at the University of Washington illuminates the relationship between big data, artificial intelligence, and machine learning in the analogy as follows. Take for example a planet that we want to inhabit for noble reasons. The space shuttle that we build to reach the planet can be taken as machine learning. The fuel in the fuel tank attached to the shuttle is the big data. And the destination is artificial intelligence (2019). Hence, machine learning is a means to achieve artificial intelligence. And the fuel that drives machine learning is big data.

During the last six decades for which AI has been around, it went through cycles of ebb and flow. From the 1980s to 1990s, AI suffered from the last bust cycle. Much of the theoretical underpinnings were difficult to materialise and the research fundings waned as the delivery and hype were discordant. These bust periods were known as the 'AI Winters' (Rebala, Ravi, and Churiwala 2019). Such winters were later overcome by the advent of machine learning and its success in solving real-world problems in classification, regression, and clustering.

With regards to machine learning's relation to data mining, Tom Mitchell provides a helpful description. He subsumes machine learning into the data mining process which has the goal of knowledge discovery from large datasets. Machine Learning is one of the important tools in data mining used to train a model that represents the dataset. Other tools and processes include database management,

data cleansing, data visualisation, and empirical regularity analysis. This data mining acts as a bridge to connect many technical areas including statistical analysis, machine learning, and informatics(1999).

3.4 Types of Machine Learning Problem

Not all problems are machine learning solvable. The problems that can be addressed by the Machine Learning algorithm is broadly categorised into three categories. This is illustrated below, along with their subcategories. in figure 5

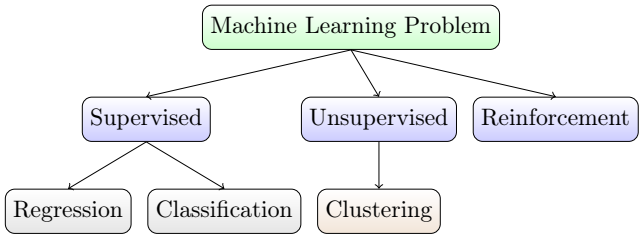


Figure 5: Types of Machine Learning Problem

3.4.1 Supervised Learning

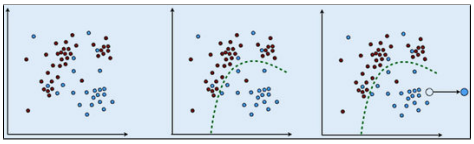


Figure 6: Supervised Learning: (1) Feature Extraction (Labelled Data) (2) Learning (Model learnt using for example the decision tree ID3 algorithm) (3) Prediction

In **supervised learning**, the machine learning algorithm is input the **dataset** of **labelled examples** where the mapping between parameter features and target features are realized. The collection of labelled examples can be stated as: $(x_i, y_i)_{i=1}^{N}$, where x is the input feature, y is the target corresponding target feature, the subscript i is the index, and N is the dataset size (Mitchell 1997: 21; Guttag 2016: 263).

Each element x_i in the N space is also known as the **feature vector**. Feature vector in which its dimension can range from $j = 1, \ldots, D$ is reasonably assumed to describe the given example. The value it holds is called a **feature** and is denoted as $x^{(j)}$. Consider, for example, a collection of students' data. If each example x in the collection represents a student, then his or her features could be stored as first feature $x_{(1)}$ could be the course enrolled in as string, the second feature $x_{(2)}$ could contain the gender as character M for male and F for female, the third $x_{(3)}$ could contain the academic year enrolled and so on. The feature vector always contains homogeneous information on feature at the position j for every example in the dataset. This is to say that, if $x_2^{(3)}$ contains the information on gender, M or F in certain example x_i, then x_k^2 will also contain the gender information in every example x_k where $k = 1, \ldots, N$.

The label y_i is the **target vector** or the information that is predicted about an example. It can be an element belonging to a finite set of classes $1, \ldots, P$ or// $\{`Malignant', `Benign'\}$ for breast tumour case, or a real number, or a complex object like a vector, tree, or a graph. A **class** is a category to which the example belongs to. For instance, if the examples are the galaxy types based on its shape then the classes can be $\{elliptical, spiral, lenticular\}$

Supervised learning algorithm has one goal which is to learn from a dataset and produce a **model** that takes feature vector x and outputs information y_i that corresponds to the feature vector. For instance, a model created using the diagnosed patient's dataset can input feature vector representing the patient such as age, tumour size, and tumour hardness and the output can be the probability of whether the person has a benign or a malignant tumour.

3.4.2 Unsupervised Learning

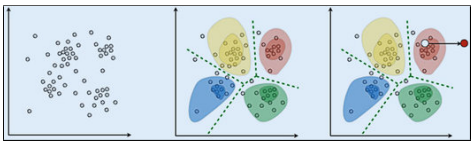


Figure 7: Unsupervised Learning: (1) Feature Extraction (Unlabelled Data) (2) Learning (Model learnt using for example the Clustering Algorithm) (3) Prediction (Model applied to new data.)

Unsupervised learning is applied to **unlabelled examples** denoted as: $\{x_i\}_1^N$. Here again, x is a feature vector. The goal of unsupervised learning algorithm is to create a model that takes in a feature vector X and outputs a transformation of input another vector or into a value that can be further used for a practical purpose (Mitchell 1997: 264); (Guttag 2016: 264).

The goal of the unsupervised learning algorithm is to discover some latent structure in the input set of feature vectors. For instance, given a set of animal feature vectors, the unsupervised learning algorithm may separate the animals into nocturnal or diurnal, perhaps into amphibians or mammals. One of the widely used supervised learning techniques is known as clustering where similar feature vectors are clustered together and partitioned. Geneticists, for instance, can employ clustering to discover groups or patterns of similar genes (Guttag 2016: 265).

3.4.3 Reinforcement Learning

Reinforcement learning employs the most unique strategy in all learning classes. The machine or **agent** is said to live in its **environment** and able to interact with it and change its **state** as a vector of features. The interaction is done in a series of **actions**. Different actions elicit different **rewards** for the agent and in the process could change its state. The goal of reinforcement learning is to discover a **policy**. A policy, similar to the model in supervised learning is a function that takes in feature vector of states and outputs an optimal i.e. maximized average reward inducing action to execute.

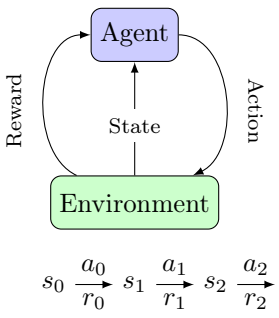


Figure 8: State Machine Representation of an Agent Interacting with its Environment.

In figure, 8 an agent lives in an environment. At an instance, it exists in a particular state s_i from a class of possible states S and performs an action a_i from a set of actions A. The agent gains real-valued reward r_i for each action executed. The overall parameters produced are a sequence of states s_i the associated each action a_i, and the instantaneous rewards for the action represented as r_i. The agent's goal is to learn an optimum control policy, $\Omega : S \rightarrow A$, that maximizes the average rewards in the sequence (Mitchell 1997: 367-370).

3.5 Ensemble Method

Our social learning behaviour is characterised by accumulating expert advice from multiple sources before making a crucial decision. The combination of myriad **'experts'** to attain **'ensemble'** decision has its roots in ancient Greece. The *Conduret Jury Theorem*, which formalises this notion during the enlightenment age, proved that the collective judgement of expert individuals are superior compared to the individual expert opinion (De Condorcet 2014). Certain techniques known as the ensemble method attempts to mimic this social learning behaviour to attain a more reliable prediction.

In machine learning, ensembles are a finite set of learning machines that combines the different learning algorithms and perspectives on a dataset to obtain more reliable and accurate predictions in supervised and unsupervised learning, in contrast to a single learning machine's performance (Dietterich 2000); (Kuncheva 2014). An example among many techniques on how this is achieved is the *majority vote* ensemble method, which employs several different learning algorithms on a dataset and maintains a record of count on each classification label for which the individual algorithm proposed true result (Cooper and Perrone 1993). The maximum count is then selected to the final true class for the prediction.

Literature Interests The literature of astronomical machine learning is rife with the alternative terms of ensemble (Drucker, Cortes, Jackel, et al. 1994; Filippi, Costa, and Pasero 1994; Sung-Bae Cho and Kim 1995; Lam and Suen 1995; Kittler, Hatef, Duin, et al. 1998; Ho 2002; Banfield, Hall, Bowyer, et al. 2007). Terms used interchangeably with the ensemble (widely recognized term in AI) include committee, fusion, aggregation, and combination. They all mean the same process of machine learning where multiple machines work together as opposed to the individual machine. In this dissertation, I have consistently used the word ensemble to mean this concept.

One of the major areas of interest within machine learning is in the field of ensemble methods (Dietterich 2000; Kuncheva 2014) This is evinced in the workshops, paper presentations, and conferences devoted to this single topic. One prominent conference is spearheaded by Kittler et al. called the Multiple Classifier Systems (MCS) (Guyer 1992; Benediktsson, Kittler, and Roli 2009; Roli, Kittler, and Windeatt 2004; Oza, Polikar, Kittler, et al. 2005; Haindl, Kittler, and Roli 2007). The efficacy of ensemble methods has been empirically studied and the experimental data against benchmark results show that they are significantly reliable than a single classifier (Banfield, Hall, Bowyer, et al. 2007; Dietterich 2000; Sohn and Shin 2007). There seem to be two major drivers for motivation for research into ensemble methods. First is, given the verified efficacy the status quo of affordable and fast computing, and powerful networks of workstations. And second is the deeper intellectual reasons to explore intrinsic characteristics of ensemble method (Way, Scargle, Ali, et al. 2012: 564).

There are also several theories proposed explaining the successful employment of ensembles. For instance, Breiman and Friedman's bias-variance analysis using the stochastic determination theory (Breiman 1996), Kleinberg's stochastic determination theory (Kleinberg 2000), and the interpretation ensemble's improved generalisations by Allwein, Schapire, and Singer in light of the large margin classifiers (Allwein, Schapire, and Singer 2001).

Ensemble method employed in Astronomy is explicated in section 3.9

3.6 The Origin of the Universe

One of the earliest determiners for the expanding universe is the result of Edwin Hubble's meticulous observations of galaxies. Redshift encapsulates the phenomena of galaxies apparently moving away and is one of the myriad significant phenomena along with cosmic microwave background radiation, that proved the phenomenon of expanding universe. In 1929, Hubble proved that distant galaxies exhibited Doppler effect as their optical spectra showed a redshift in the emitted light (1929). This was a clear indication that galaxies were moving away from the point of sight and in-fact every cosmic object was flying apart as the space between them stretched. If one were to rewind the cosmic movie of expanding universe, then there would come a point when everything would collapse back at a singular point known as the singularity which is the boundary of space, time, matter, and energy (Craig 1999; Davies 2007: 18).

3.7 Redshift Measurements

One of the frustrations of an observational science like astronomy is we can't set up experiments in the lab. It is not possible to control the unknown parameters to test hypotheses or use physical equipment like rulers to measure

distances. Instead, a cleverly constructed observations are required to collect data that will help answer such questions. A redshift defectively gives us a distance to a galaxy and so by doing this, we can map out the universe in 3D(Filippenko 2007). Measuring spectroscopic redshift is the most accurate way of doing this but there are more galaxies for which imaging observations over spectroscopic ones are available. So a set of galaxies with known redshifts used for training the machine learning classifier to calculate redshifts for new sets of galaxies. This is the type of problem that machine learning is perfect for (Ivezić, Connolly, VanderPlas, et al. 2014; Way, Scargle, Ali, et al. 2012). This is the task where astronomical elements are well understood, but it is not possible to do it with a rule-based approach.

Machine learning is a fast solution that allows us to evaluate the accuracy of the results and work out the probability of the results being correct. It can often seem like there's lots of hidden voodoo going on in machine learning. But really all we're doing is building a model based on known data and then applying that model to unknown data. In this study, I am able to calculate redshifts for the hundreds of thousands of galaxies. effectively using them as a tool to measure distance in the universe. But before the actual mechanics, theoretical foundations and the literature needs to be layed out.

It is quite straightforward to calculate the redshift of the galaxy. In the presence of spectral fingerprint of elements in the earth that are also found in galaxies, the redshift is obtained by subtracting the elemental spectrograph data into the observed galaxy's spectrograph data. an example is shown in figure 12. However, the process can be time-consuming and tedious for large samples. Likewise, without the reference and when the only spectrum for the galaxy is available it can be difficult to estimate the photometric shifts. This is where machine learning can help. Given the accurately calculated spectrum for other galaxies, the ML model is able to predict the redshifts in new galaxies.

Five decades ago Baum proposed the first measurement scheme for spectral energy distribution involving six elliptical galaxies. Uncharted territories were traversed for the first time when he later used photometry instead of spectroscopy to measure the redshifts (Way, Scargle, Ali, et al. 2012: 325).

3.8 Galaxy Classification

The 18th-century philosopher Emmanuel Kant marvelled at what he called 'the moral law within and the starry heavens above' (Kant 2012: 1). The former aspect in the dictum is encapsulated in his noted discourses in epistemology, ethics, and metaphysics.

The Nebulae Debate While the later which is lesser-known aspect is evinced in his theoretical studies in Astronomy. During the 1920s, a debate raged between astronomers about the massive fuzzy accretion in the night sky called the 'nebulae'. Some believed they were clouds of local stars within the milky way, while others contested that they were extragalactic star systems. The latter camp was proposed centuries before the debate and before the existence of telescopes, by Hubble (Hubble, Kirshner, and Carroll 2013).

Classical Classification Putting the nebulae debate to perpetual rest, Hubble had devised the first classification scheme for galaxies based on their morphology (1929). It was based on the conspicuous physical feature of the galaxies observed. The ones with the bulge were labelled 'Early-Type Galaxies (ETG)', whereas the others with the prominent disc were labelled 'Late-Type Galaxies (LTG)'. His classification scheme is shown below in figure 9.

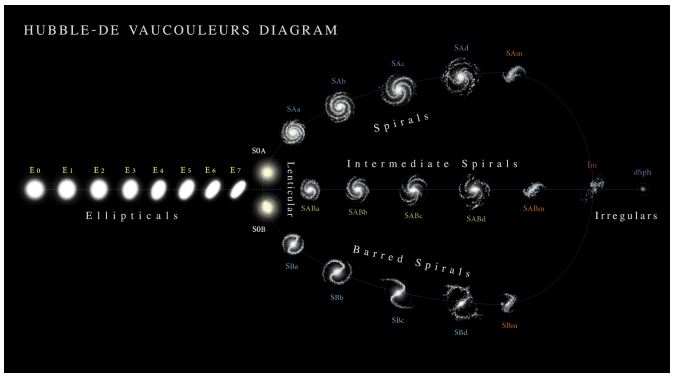


Figure 9: Hubble-De Vaucouleurs Morphological Classification Scheme of Galaxies

The spiral galaxies are further forked into barred (SB) and non-barred (S). De Vaucouleurs further refined the classification by adding T-Type numbers to further divide the spirals and ellipticals based on their spirality and ellipticity strengths respectively. All modern forms of galaxy classification schemes are the extended forms of the Hubble Sequence.

Evolution Misconception One of the popular misconceptions about the Hubble Sequence is that it depicts an evolutionary sequence of galaxies from the ellipticals on the left evolving to morph into one with a disk. However, Hubble, himself in his seminal work advised that 'temporal connotation is made at one's peril' and further he explains his scheme is without prejudice of the evolution of galaxies (Hubble, Kirshner, and Carroll 2013: 277).

Since Hubble's classification scheme, there have been numerous take on updating it (Buta 2013). But, the core of the scheme remains intact with only minor changes. What has significantly changed is the acceleration in the number of galaxies classified using the scheme and still a massive number of galaxies yet to be classified. Following at the footsteps of Hubble, and with the transition of astronomers from visual photographic plate inspection to modern powerful telescopes and detectors, detailed morphologies have been produced and the results published. Significant catalogues include, The Hubble Atlas of Galaxies (Sandage 1984)and the Third Reference Catalogue of Bright Galaxies (RC3) (Vaucouleurs 1991). These and other catalogues are available at and maintained by the NASA/IPAC Extragalactic Database (Jet Propulsion Laboratory 2019).

Imperative for Machine Learning Approach The problem with expert classification is the implausibility and infeasibility of classification due to the sheer size of the galaxy dataset. Consider for example the galaxy data collected by the Sloan Digital Sky Survey (SDSS). It exceeds

in the figures of millions and cannot be visually inspected even by the ensemble of expert astronomers at work concurrently (Strauss, Weinberg, Lupton, et al. 2002; York, Adelman, Anderson, et al. 2000). It was conspicuous that an automated classifier system was required to solve this issue. However, traditional computer programming would not be able to take account the complexities of galaxies' shapes such as the differences in attributes such as spiralities, bars, bulges, brightness and disks (Way, Scargle, Ali, et al. 2012: 216).

Automated Classification Proposals One of the earliest automated galaxy classification scheme according to their morphology can be traced back to the proceedings of the astronomy conference entitled: 'Le Monde des Galaxies' (The World Of Galaxies) which was held in 1989, April 12 -14 in the Institut d'astrophysique de Paris (Paris Institute of Astrophysics) (Thonnat 1989). The synopsis of the paper is depicted below in figure 10.

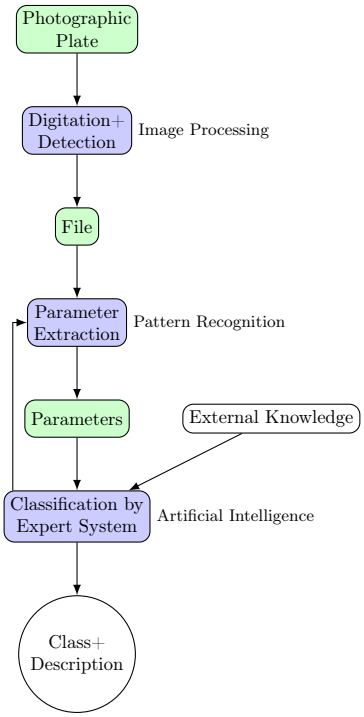


Figure 10: Thonnat's synopsis of automated morphological classification. (Thonnat 1989: 54)

Barchi et al. cite the importance of morphological classification of galaxies to understand the large-scale structure of the Universe (Barchi, Carvalho, Rosa, et al. 2020). They categorize decision trees to traditional approach contrasting the more advanced, deep learning technique. One may assume that the deep learning technique would yield more accurate predictions. However, the 94.5% accuracy is attributed to both the approaches. The focus of my research is training the decision tree and ensemble classifiers. Since the choice of model for advanced techniques yields approximately the same results, I have selected the binary classifier which is more intuitive to understand and the underlying concepts of machine learning are better illuminated. Barchi et al. only classify elliptical and spiral galaxies. (Barchi, Carvalho, Rosa, et al. 2020). In contrast, I add and classify, the 'Merger' type galaxies which are the conglomeration of two galaxies colliding into each other.

Another early attempt at automated classification is due to Lahav et al. They compared the classification of 830 galaxies independently classified prior to, by six human classifiers (Lahav, Naim, Buta, et al. 1995). It was concluded that there was unanimous agreement to merely less than 1% of objects and at best 80% agreement on T-Type galaxies. It was evident that parameters such as colour, image quality, and size are crucial for the classification and that artificial neural networks could mimic the human experts. This was demonstrated by Ball et al. on the SDSS dataset (2004) with similar conclusions to Lahav et al (1995).

3.9 Ensemble Methods in Astronomical Machine Learning

Astronomy has been one of the first disciplines to witness the influx of exponential amount of data. A common problem encountered by astronomers as Valentini and Re note is the classification of a vast array of celestial objects according to an established classification scheme (2012: 579). For instance, in the case of galaxy classification according to morphology, the Hubble tuning fork is an important starting point. Moreover, another problem in classification is to discover embedded relations among unlabelled or uncategorised objects. Ensemble methods are regarded as a tool rather than off-the-shelf solutions to solve these problems (Way, Scargle, Ali, et al. 2012: 579).

A good example of automated annotation using ensemble methods is the classification of galaxies based on their morphology. Bazell and Aha compared three classifiers namely, Naive Bayes, decision tree, and a neural network that were tasked to predict 800 galaxies (2001). 10-fold cross-validation was performed and average classification errors were collected. The authors observe that the error reduction rate is inversely proportional to the number of target classes, and it varies according to the classifier chosen. Valentini & Re states the Italian computer scientists state that it is impossible to determine *a priori*, the best classifier component to select for a problem due to the nature of the problem and the nature of the dataset at hand (2012: 579). Therefore, they maintain that it is imperative to perform an experimental analysis with different base classifiers to determine their performance. For the morphological classification of galaxies. I have chosen Random Forest as ensemble classifier and assessed its performance against decision tree and with regards to the true classifications.

4 Methodology

"It doesn't stop becoming magic just because you know how it works."

Sir Terry Pratchett
Discworld, 1983

4.1 Overview

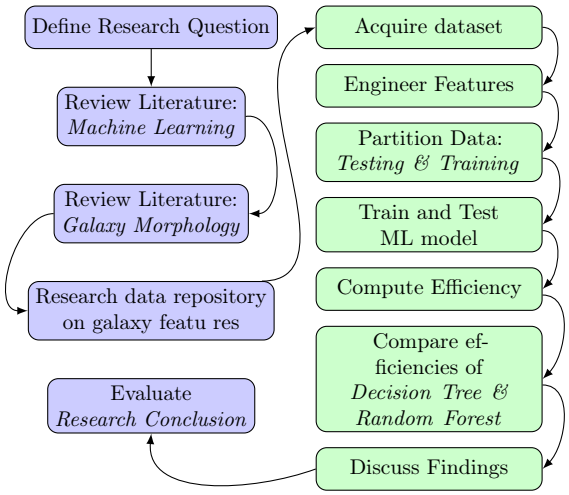


Figure 11: Research Methodology that I have followed.

4.2 Regression Classifiers

Before exploring the classification, a decision tree is first applied to the regression problem. The following sections deal with the determination of redshifts of galaxies from their photometric colours. It is independent of the morphological classification of galaxy however the flux data of both the problems are shared by the galaxy. Hence the same data can be used to predict both redshift and morphology. Although the colour profiles across 5 SDSS[2] filters[3] Thus flux data across 5 bands are one of the most significant features in this project. are present in both the problems. I have deliberately included the regression problem before tackling the morphological classification in order to demonstrate how decision tree can work for both classifications and for regression problems and to complement classification with the learnings from regression. Moreover, the difference in terms of the way the efficiency of the decision tree is assessed in each case is also significant.

4.2.1 Photometric Redshift

Since time immemorial, the night sky has evoked wonder in the heart of man and his mind has ever since been restless to discern the mysteries of the universe. The existential questions of origin have intrigued philosophers and cosmologists exchanging insights into the nature of the universe. Surpassing the steady-state model, modern cosmology has now discovered multiple independent verifications (such as the cosmic microwave background radiation) for the beginning of the universe (Ross 2008). The standard model now sets the age of the universe at approximately 13.7 billion, whence it was born in a cataclysmic event known as the Big Bang. Since then the universe has ever been expanding.

Galaxy Redshift & Big Bang One of the earliest determiners for the expanding universe is the result of Edwin Hubble's meticulous observations of galaxies. In 1929, Hubble proved that distant galaxies exhibited Doppler effect as their optical spectra showed a redshift in the emitted light (1929). This was a clear indication that galaxies were moving away from the point of sight and in-fact every cosmic object was flying apart as the space between them stretched. If one were to rewind the cosmic movie of expanding universe, then there would come a point when everything would collapse back at a singular point known as the singularity which is the boundary of space, time, matter, and energy (Davies 2007: 18).

Redshift is a significant phenomenon in astronomy that proved the expanding universe. In the presence of spectral fingerprint of elements in the earth that are also found in galaxies, it is quite straightforward to calculate the redshift of the galaxy. The redshift is obtained by subtracting the elemental spectrograph data into the observed galaxy's spectrograph data. an example is shown in figure 12. However, the process can be time-consuming and tedious for large samples. Likewise, without the reference and when the only spectrum for the galaxy is available it can be difficult to estimate the photometric shifts. This is where machine learning can help. Given the accurately calculated spectrum for other galaxies, the ML model is able to predict the redshifts in new galaxies.

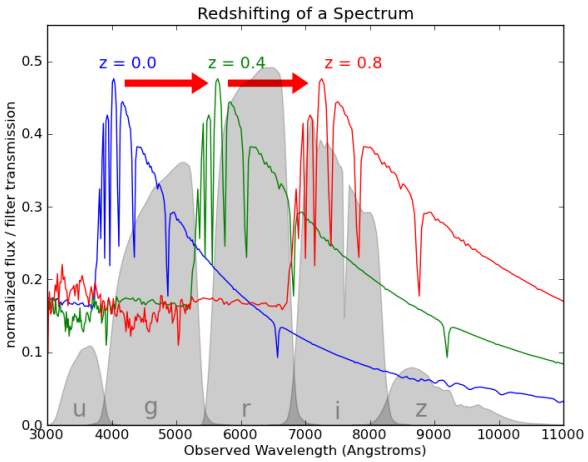


Figure 12: SDSS Filters, Reference Spectrum, and Redshifting Spectrum(ref image sdss).

4.2.2 Magnitudes and Colours

The features used in the training dataset comprises of the flux magnitudes of the Sloan Digital Sky Survey (SDSS) catalogue to create the colour index. The flux magnitude is the intensity of light or total light received in five frequency bands - u, g, r, i, and v through the SDSS filters.

[2]The SDSS is one of the longest-running astronomical observation that scans the multi-spectral redshifts of galactic matter. It employs a 2.5m wide-angle telescope housed at the Apache Point Observatory in New Mexico, USA (Sloan Digital Sky Survey 2014)

[3]Light is the goldmine of information used by astronomers to, deduce for example a star or a galaxy's distance, composition, mass, temperature and other attributes. And colours are the functions of different wavelength of light. Here, to predict the type of galaxy one of the feature is colour. Astronomers have precisely defined colour to accommodate all its complexities devoid of any subjectivity. Colour is the difference of luminous flux/brightness of source between two filters. SDSS uses 5 filters - u(ultra-violet), g(green), r(ed), i(near-infrared), and z(infrared). Infrareds are invisible to the human eye. A galaxy with high g-r (difference) colour is redder than one with low g-r colour.

4.2.3 Decision Trees in Python

In this prediction, the decision colour indices from photometric imaging will be the input and the photometric redshift will be the output. In the decision tree, the colour indices are the features at the top nodes and the target label - the redshift value which is continuous is reaches the leaf nodes at the bottom. This is depicted in the tree shown in figure 13

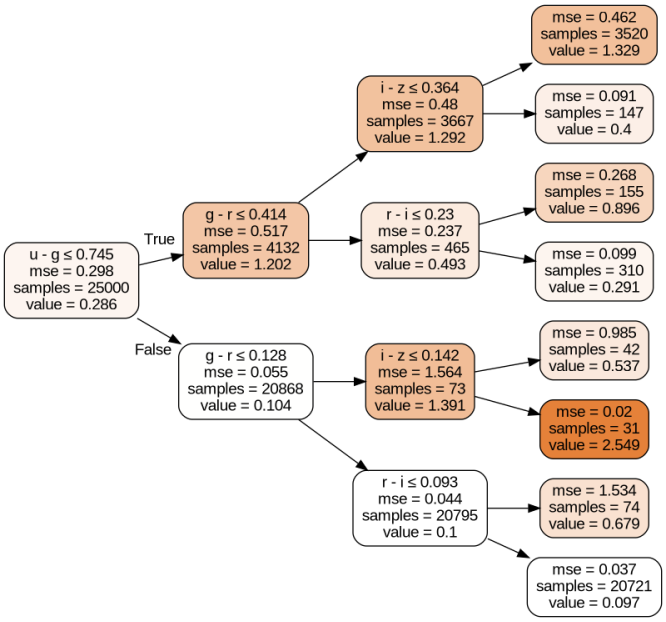


Figure 13: Graphic view of decision tree generated in Python for the redshift regression.

The python machine learning library `scikit-learn` is used as the regression classifier. I take in the input features along with the corresponding target values, then constructs a tree model that it employs to predict new data.

4.2.4 The Sloan Data

The Sloan data is a `NumPy` object which is a structured array. The `NumPy` package in Python is a fundamental tool for scientific computing. It provides features such as powerful multi-dimensional array with useful manipulation functions (). The colour indices and redshift data is stored as shown in table 2

u	g	r	i	z	...	redshift
19.84	19.53	19.47	19.18	19.11	...	0.54
19.86	18.66	17.84	17.39	17.14	...	0.16
...	...	...	...	...	...	
18.00	17.81	17.77	17.73	17.73	...	0.47

Table 2: A sample of content in the NumPy record - Flux magnitudes across 5 colour indices with redshift value.

The ellipsis in between hints at additional data and also the presence of columns 'spec_class' and 'redshift_err' which is not required in this case.

4.2.5 Features & Targets

Extraction of features and target definition is written in the function `get_features_targets` shown in figure 14

```
def get_features_targets(data):
  #n rows, 4 columns
  features = np.zeros((data.shape[0], 4))
  features[:,0] = data['u'] - data['g']
  features[:,1] = data['g'] - data['r']
  features[:,2] = data['r'] - data['i']
  features[:,3] = data['i'] - data['z']

  targets = data['redshift']

  return (features, targets)
```

Figure 14: Subroutine to extract features and to define the target.

It splits the training data into input features and the respective targets. In this case, the 4 colour indices and the respective redshifts as targets. It returns a tuple of:

- **features**: a `NumPy` array of n by 4 dimensions, where n is the number of galaxies or the dataset size.
- **targets**: a 1-dimensional `NumPy` array of size n, containing the corresponding redshift value for each galaxy.

Specific columns can be accessed with the index name. For example, the 'z' flux magnitude and its redshift can be accessed as `data['u']` and `data['redshift']` respectively. The four features are the colour difference 'u-g','g-r','r-i',i-that have been dynamically calculated from existing columns.

4.2.6 Limitations of Non-ML approach

The redshift prediction based on colour index could be approached by a non-machine-learning approach. An empirical model could be constructed to predict the change in redshift value (dependent variable) with uni or multivariate colour indices (independent variable).

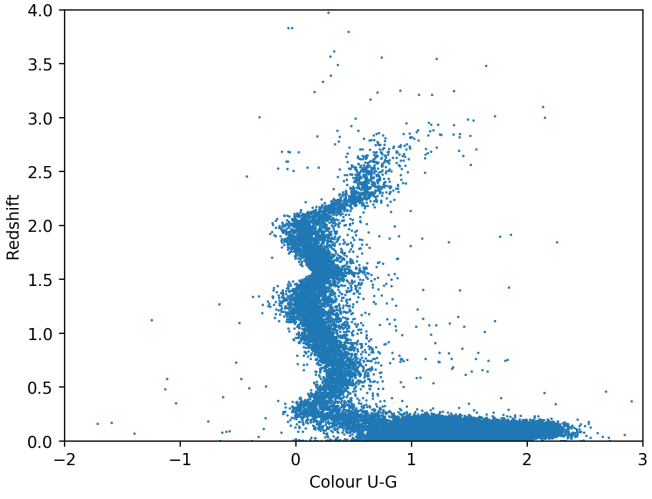


Figure 15: Redshift vs Colour index 'u-g'.

For example, in the graph above a non-linear regression function could be constructed. Least mean squares could

be used to determine the best fit. It can be predicted from the spread of values that this model would be a complex one such as an inverse sine function. The model obtained could be used. However, it would turn out to be highly complex and there would be no assurance that it would yield reliable results.

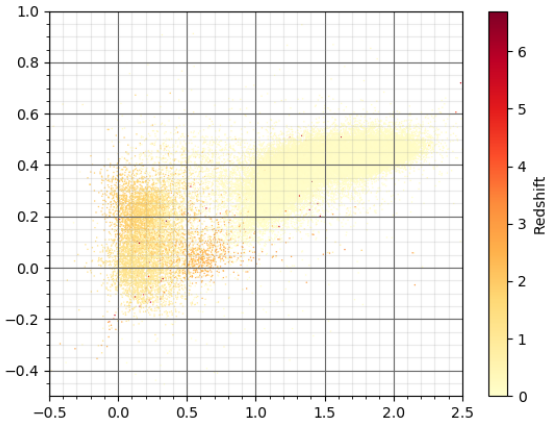


Figure 16: Redshift vs two colour indices.

Another approach could be to graph colour-index vs colour-index plot using the additional bands to depict the redshift. An example of this is shown above in figure 16. The two indices show certain defined regions where redshifts are similar. A combination of other indices could be used and plotted on a contour map. However, the resulting redshift estimates would have significant uncertainties.

4.2.7 Decision Tree Regressor

The readied features and targets needs to fed to the decision tree model so that it can learn before making the predictions. The `DecisionTreeRegressor` class from `sklearn.tree` is used for this. The algorithm is initialised by the statement `drt=DecisionTreeRegressor()`. To train the model, we use the `fit` method while passing the parameters `features` and `targets` as: `dtr.fit(features, targets)`. The `dtr` object now trained can be used to make the prediction. This is achieved by the statement: `predictions=dtr.predict(features)`. Here, `predictions` is an array of predicted redshift values mapping to each of the galaxy. The first 4 predictions are seen below in figure 17

```python
import numpy as np
from sklearn.tree import DecisionTreeRegressor

def get_features_targets(data):
  #n lines, 4 columns
  features = np.zeros((data.shape[0], 4))
  features[:,0] = data['u'] - data['g']
  features[:,1] = data['g'] - data['r']
  features[:,2] = data['r'] - data['i']
  features[:,3] = data['i'] - data['z']
  targets = data['redshift']
  return (features, targets)

# loading the data and
data = np.load('sdss_galaxy_colors.npy')
# generating the features and targets
features, targets = get_features_targets(data)

# initializing model
dtr = DecisionTreeRegressor()

# training the model
dtr.fit(features, targets)

# making predictions using the same features
predictions = dtr.predict(features)

# printing the first 4 predicted redshifts
print(predictions[:4])
```

Figure 17: Method - Decision Tree Regression Prediction.

The output is the first 4 predictions is seen below in figure 18:

```
[biju@biju-pc regression]$ python 02.decision_tree_regressor.py
[0.539301   0.1645703  0.04190006 0.04427702]
```

Figure 18: First 4 predictions of redshifts.

4.2.8 Evaluating The Accuracy

```python
import numpy as np

# calculating the median of the differences
# between our predicted and actual values
def median_diff(predicted, actual):
  diff = np.median(np.absolute(predicted - actual))
  return diff


if __name__ == "__main__":
  # load testing data
  targets = np.load('targets.npy')
  predictions = np.load('predictions.npy')

  # calling the function to measure
  #  the accuracy of the predictions
  diff = median_diff(predictions, targets)

  # print the median difference
  print("Median difference: {:0.3f}".format(diff))
```

Figure 19: Code to calculate the accuracy using the median difference.

To ensure the accurate prediction in regression, the predictions generated by the model and the actual values are compared to test the accuracy of the model and thus its performance. The difference between the actual and predicted value is the residual. It can give a preliminary assessment of the model's performance. There are multiple methods for residual (Liitiäinen, Verleysen, Corona, et al. 2009), where the median of differences of predicted and

actual value is used. It is calculated as:

$$A = M(|Y_{i,predicted} - Y_{i_actual}|)$$

where A = accuracy,

M = median,

$||$ = absolute value of the difference

To achieve the median difference in code, the following method shown in figure 20is prepared:

```
import numpy as np

# calculating the median of the differences
# between our predicted and actual values
def median_diff(predicted, actual):
  diff = np.median(np.absolute(predicted - actual))
  return diff


if __name__ == "__main__":
  # load testing data
  targets = np.load('targets.npy')
  predictions = np.load('predictions.npy')

  # calling the function to measure
  #  the accuracy of the predictions
  diff = median_diff(predictions, targets)

  # print the median difference
  print("Median difference: {:0.3f}".format(diff))
```

Figure 20: Subroutine to compute the median difference.

4.2.9 Hold-out Validation

The decision tree regression shown above used the common dataset for both training and testing the decision tree. This is only the optimisation of the model to get the best results for only training data. Hence, it does not provide a realistic estimate for accuracy for unseen galaxies. The way to solve the problem is to split the dataset into two subsets - training and testing. This method of validation is known as the hold-out method (Schneider 2019). The corresponding python `method validate_model` which receives model, features, and targets as input arguments, is shown below in figure 21

```
import numpy as np

# calculating the median of the differences
# between our predicted and actual values
def median_diff(predicted, actual):
  diff = np.median(np.absolute(predicted - actual))
  return diff


if __name__ == "__main__":
  # load testing data
  targets = np.load('targets.npy')
  predictions = np.load('predictions.npy')

  # calling the function to measure
  #  the accuracy of the predictions
  diff = median_diff(predictions, targets)

  # print the median difference
  print("Median difference: {:0.3f}".format(diff))
```

Figure 21: Subroutine for hold-out validation.

Hold-out validation method invocation is shown below in figure 22. And the output is shown in figure 23

```
if __name__ == "__main__":
  data = np.load('sdss_galaxy_colors.npy')
  features, targets = get_features_targets(data)

  # initialize model
  dtr = DecisionTreeRegressor()

  # validate the model and print the med_diff
  diff = validate_model(dtr, features, targets)
  print('Median difference: {:f}'.format(diff))
```

Figure 22: Hold-out validation method invocation

```
[biju@biju-pc regression]$ python 04.validating_our_model.py
Median difference: 0.021889
```

Figure 23: Median difference after Hold-out validation.

As shown in figure 23 the median of differences resulted resulted to ≈ 0.02. Since the dataset was split in half, this suggests that half of our galaxies have residual error in the prediction of < 0.02, which is significantly low and considerable.

4.2.10 The Problem of Overfitting Tendency

Decision trees have a tendency to overfit the data. This means that they try to accommodate for the extreme values at the cost of prediction accuracy of the general model. The held-out method is insufficient for overcoming this problem. A more robust method would be the k-fold cross-validation. The cross-validation is performed k a number of times and each time the partition size of training and testing sets vary, then the results are averaged (Mitchell 1997).

To visualise overfitting, the tree performance first needs to be examined at various tree depths. Naïvely, it may be assumed that the greater the tree depth, the better it performs. However, this is not the case as when the model overfits the accuracy on the testing data reduces although training data may show high accuracy.

```
import numpy as np

# calculating the median of the differences
# between our predicted and actual values
def median_diff(predicted, actual):
  diff = np.median(np.absolute(predicted - actual))
  return diff


if __name__ == "__main__":
  # load testing data
  targets = np.load('targets.npy')
  predictions = np.load('predictions.npy')

  # calling the function to measure
  #  the accuracy of the predictions
  diff = median_diff(predictions, targets)

  # print the median difference
  print("Median difference: {:0.3f}".format(diff))
```

Figure 24: Subroutine to return accuracy by tree depth.

To control the tree depth, the statement `dtr = DecisionTreeRegressor(max_depth=5)` is used. The method `accuracy_by_treedepth` shown above in figure 24returns the median difference for both

the testing and the training sets looping for each depth in the `depths` value. It expects the folllowing parameters:

- `features` and `targets` : same as previous reference
- `depths` : an array of depth to be used for decision tree regressor's `max_depth` argument.

The method returns two array lists `mediandiffs_training = []` and `mediandiffs_test = []` containing the `median_diff` values for predictions made using only the training set and for predictions using only the testing set; along with the depths. For instance, if `depths` is `[5,7,9]` then the method returns two lists of size 3.

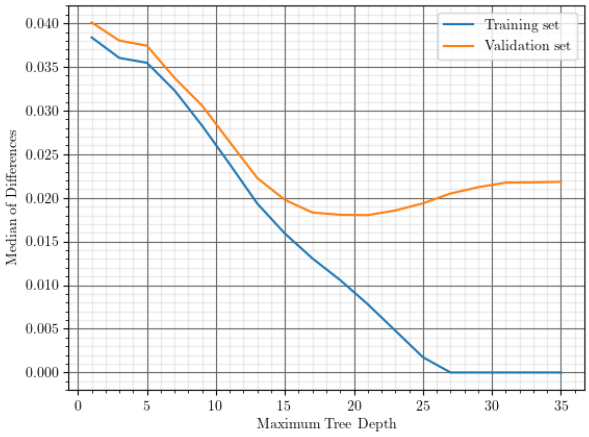


Figure 25: Overfitting problem in decision tree visualised.

In figure 25, it is clearly seen that the accuracy of the training set improves as the tree grows. However model from the validation set starts to overfit after 19^{th} tree depth. Overfitting, a common problem with decision tree can be mitigated by tuning the tree depth parameter. In this case for the redshift problem, the minimum depth of 19 can be adjusted. The overfitting problem based on figure 25method is further discussed in the finding and discussion section in 5.1.3.

4.2.11 *K*-fold Cross Validation

Hold-out validation where the dataset is partitioned to training and validation set is a basic one and better than no validation. However, its accuracy or the median of differences will vary depending upon how the split was done. It would have no experience in predicting the outliers. To optimise its experience, k-fold cross-validation can be used. Similar to the holdout validation, the dataset is split but this time to k subsets or folds. The training and testing occur k times, and each time the accuracy is recorded. For each iteration, a unique k-1 subset is used for training and the index k subset for testing. Finally, the average of the k measurements is taken to be the robust accuracy of the model.

The method to perform the k-fold cross validation is defined as `cross_validate_predictions`, which is shown below in figure 26.

Figure 27: Anatomy of a Quasar
(Credit: NASA/Goddard Space Flight Center Conceptual Image Lab 2008)

```
def cross_validate_model(model, features, targets, k):
  kf = KFold(n_splits=k, shuffle=True)

  # initialise a list to collect
  # median_diffs for each iteration of the loop below
  mediandiffs = []

  for train_indices, test_indices in kf.split(features):
    train_features, test_features =
    features[train_indices], features[test_indices]
    train_targets, test_targets =
     targets[train_indices], targets[test_indices]

    # fit the model for the current set
    model.fit(train_features, train_targets)

    # predict using the model
    predictions = model.predict(test_features)

    # calculate the median_diff from
    #  predicted values and append to results array
    mediandiffs.append(median_diff(test_targets, predictions))

  # return the list with your median difference values
  return mediandiffs
```

Figure 26: Subroutine for k-fold cross validation.

Python provides the `KFold` library for the k-fold cross validation. It provides an iterable object that is initialised, for example with the statement: `kf=KFold(n_splits=k, shuffle=True` The `n_splits=k` specifies the number of splits to perform. To ensure there was no order, the shuffle value is toggled to true.

4.2.12 Quasars vs Galaxies

A quasar also known as a quasi-stellar object (QSO) are a type of galaxy that contains a core of accreting supermassive blackhole. Thus it is said to have an active galactic nucleus. It differs from regular class of galaxies in that the gas surrounding its accretion disk is actively collapsed in the central black hole and altogether it radiates tremendous amount of energy (NASA 2016). Wu et al. reports the discovery of a quasar labelled 'SDSS J010013.02+280225.8', whose luminosity exceeds thousands of times compared to the Milky Way galaxy (Wu, Wang, Fan, et al. 2015).

By definition, quasars are intensely brighter than the rest of the type of galaxies. And their detection will be the result of higher redshift values. In the dataset used so far,

the redshift for galaxies has been in the range of $z \approx 0.4$, while the redshifts of quasars can exceed the magnitude $z \approx 6$. The accuracy of the decision tree between QSOs and regular galaxies can vary.

```
def split_galaxies_qsos(data):
  # splitting the data into galaxies and qsos arrays
  galaxies = data[data['spec_class'] == b'GALAXY']
  qsos = data[data['spec_class'] == b'QSO']

  # returning the seperated galaxies and qsos arrays
  return (galaxies, qsos)

def cross_validate_median_diff(data):
  features, targets = get_features_targets(data)
  dtr = DecisionTreeRegressor(max_depth=19)
  return np.mean(cross_validate_model(dtr, features, targets, 10))
```

Figure 28: Method to separate dataset based on spectral class.

The method `split_galaxies_qsos` shown in figure 28 separates the dataset based on the type i.e. galaxies and QSOs. It returns two `NumPy` arrays containing the dataset with galaxies and another containing the dataset with quasars. It does so based on the column `data['spec_class']` where the value spectral class is stored in byte String format which is either `b'GALAXY'` or `b'QSO'`. The inner `data['spec_class'] == b'GALAXY'` returns all of the indices that have a galaxy spectral type. These indices are then used to select the rows with the outer `data[...]`

The complete is *

4.3 Classification

4.3.1 Classifier's Gold Standard - Galaxy Zoo

Galaxy Zoo is a citizen science or crowdsource based non-commercial and a purely scientific astronomy project (Raddick, Bracey, Gay, et al. 2010; Raddick, Bracey, Gay, et al. 2013). The idea is to leverage public participation to accelerate scientific research. The goal is to assist the astronomy community in the morphological classification of galaxies. Since the past decade, the project has produced myriad iterations of visual classifications and subsequent data releases for hundreds of thousands of galaxies.

The dataset used in the subsequent training is obtained from Galaxy Zoo which is the result of the Sloan Digital Sky Survey.

Figure 29: Spiral, Elliptical, and & Merger Galaxies (From left, Messier 101 (NASA/JPL Caltech 2017), ESO 325-G004(Space Telescope Science Institute 2007), & Arp 240 - merger of NGC 5257 and NGC 5258 (European Space Agency 2019)

The dataset is limited to three types of galaxies as shown in figure 29. At least 20 human classifiers have a common consensus on the results. For type elliptical and spiral there is a unanimous classification. And due to low sample numbers, merger class was included as 80% of the human classifiers have agreed to its class (Lintott, Schawinski, Slosar, et al. 2008; Lintott, Schawinski, Bamford, et al. 2011). This dataset is used to classify train the classifiers.

4.3.2 Deciding upon the Features

While automated feature decision techniques using raw pixels of images are available in neural networks and deep learning, machine learning, especially in astronomy, require an expert to design the feature set (stat astro book). Here, known galaxy profiles are used ln the basis of SDSS's flux magnitude data on 5 bands (u, g, r, i, z) filters.

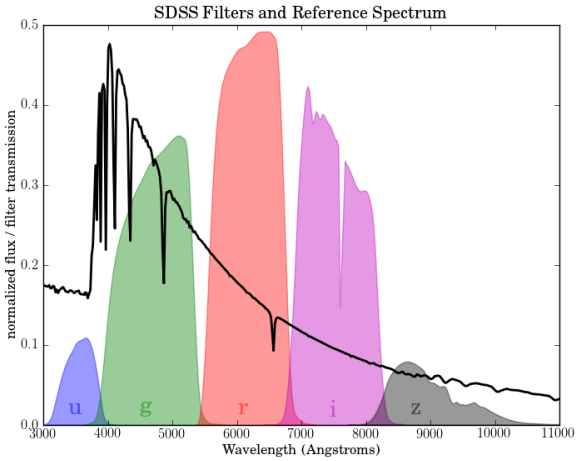


Figure 30: Flux magnitude across 5 SDSS filters (ref image sdss).

Following features are selected for the morphological classification of galaxies. These features are a subset of the SDSS catalogue of features.

- **Color Indices** are the differences in color pairs - 'u-g', 'g-r', 'r-i', and 'i-z'. Studies of galaxy evolution show that spiral galaxy comprises of younger stars and thus are bluer i.e. brighter at lower wavelengths. While elliptical galaxies have older stars and are redder, brighter at the lower wavelengths.
- **Eccentricity** is the approximation for the shape of the galaxy measured as a ratio of the major and minor axis of the ellipse imposed to the galaxy. Galaxy Zoo uses the De Vaucouleurs model to attain these axes. (Bartola) Here, the median eccentricities of 5 filters are used.

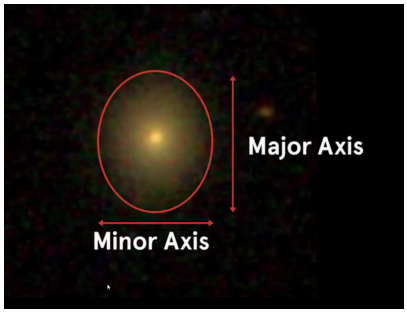


Figure 31: Example of ellipse imposed on an elliptical galaxy.

- **Adaptive Moments** is another marker for the shape of the galaxy. These are mostly used in image analysis to discern similar objects despite their size and orientations. The dataset has the 4^{th} order moment for each band.

- **Concentration** is a more granular luminosity profile compared to the colour indices. These measures which radius section of the galaxy (when imposed with an annulus) produces which proportion of total emitted light. A way to represent the concentration is to take the ratio of 50% and 90% of the Petrosian flux[4]

$$C = \frac{P_{R1}}{P_{R2}}$$
$$\text{where } C = \text{Concentration},$$
$$P_{R1} = 50\% \text{ petrosian flux},$$
$$P_{R2} = 90\% \text{ petrosian flux}$$

4.3.3 Describing the Dataset

```
[biju@biju-pc Py]$ python 01_split.py
Number of galaxies in dataset: 780
Train fraction: 0.7
Number of galaxies in training set: 546
Number of galaxies in testing set: 234
Field and value for 1 Galaxy:
u-g                 1.60844
g-r                 0.75164
r-i                 0.38429
i-z                 0.2769
ecc                 0.88749
m4_u                2.35123
m4_g                2.41662
m4_r                2.49306
m4_i                2.54
m4_z                2.5764
petroR50_u          8.29955
petroR50_r          9.05789
petroR50_z          8.19888
petroR90_u          16.8653
petroR90_r          19.7562
petroR90_z          18.447
class               spiral
[biju@biju-pc Py]$
```

Figure 32: Metadata of NumPy Record of Galaxy data.

The dataset is contained in a `NumPy` file of 780 records. Figure 32 shows 4 color features, the median eccentricity, adaptive moment at each filter, the Petrosian fluxes for each filter, and lastly the class or the type of the galaxy these features belong to.

4.3.4 Splitting into the Training and the Testing Set

The classification first begins by splitting the dataset into training and testing portions. Here, a function is defined to perform the splitting. It requires two parameters:

- **data**: The `NumPy` dataset as described above.

- **fraction_training**: The fraction of the dataset to be used for the training set. It ranges from 0 to 1. The function `math.floor()` keeps in check if the fraction splits the record into a decimal number. It does so by truncating the decimal part. For example, $0.67 * 780 = 722.6$ training rows would be truncated to 722.

The Return objects of the function are:

- training_set `NumPy` array of training set.

- testing_set: `NumPy` array of testing set.

It is a good practice to shuffle the data to disrupt possible order from prior extraction. It ensures balanced class of targets are realized by the training algorithm. The statement `np.random.seed(0)` is used for consistent shuffling pattern and `np.random.shuffle(data)'` is used to perform the shuffling on the data.

4.3.5 Generating Features and Targets

```
def generate_features_targets(data):

  targets = data['class']
  #Create array where.
  #row = no. of trainng set record, column = 13
  features = np.empty(shape=(len(data), 13))

  features[:, 0] = data['u-g']
  features[:, 1] = data['g-r']
  features[:, 2] = data['r-i']
  features[:, 3] = data['i-z']
  features[:, 4] = data['ecc']
  features[:, 5] = data['m4_u']
  features[:, 6] = data['m4_g']
  features[:, 7] = data['m4_r']
  features[:, 8] = data['m4_i']
  features[:, 9] = data['m4_z']

  # filling the remaining 3 columns with
  # concentrations in the u, r and z filters
  features[:, 10] = data['petroR50_u']/data['petroR90_u']
  features[:, 11] = data['petroR50_r']/data['petroR90_r']
  features[:, 12] = data['petroR50_z']/data['petroR90_z']

  return features, targets
```

Figure 33: Subroutine to generate features and targets set.

The function shown in figure 33 generates the features and target. It works for both training and testing set and returns two `NumPy` arrays of features and targets. The last three columns are reserved for Petrosian flux measurements. This is calculated using the equation $conc = \frac{petro_{R50}}{petro_{R90}}$.

So the statement,

```
features[:, 12]= \ data['petroR50_z']/data['
↪ petroR90_z']
```

calculates the Petrosian flux for z filter and loads it into the 13^{th} column.

Now, we have the following set of features and respective target class ready:

- **colours**: u-g, g-r, r-i, and i-z

- **eccentricity**: ecc

[4]The Petrosian flux is one of the most significant measures in astronomical photometry. Photometric flux measurement for galaxies is harder compared to stars because they have no prominent edge and the brightness distribution in its body differ radially. In order to remove bias and measure a constant portion of total emitted light, which is independent of the galaxy's position or distance, Petrosian flux is used. SDSS uses a modified form of Petrosian where azimuthally (annular) averaged light profile is measured within the circular frame (Sloan Digital Sky Survey 2019) The concentrations from u, r,z filters are used in the dataset.

- 4^{th} adaptive moments: m4_u, m4_g, m4_r, m4_i, and m4_z
- **50% Petrosian**: petroR50_u, petroR50_r, petroR50_z
- **90% Petrosian**: petroR90_u, petroR90_r, petroR90_z

4.3.6 Training the Decision Tree

```
def dtc_predict_actual(data):
  # split the data into training and
  # testing sets using a training fraction of 0.7
  training_set, testing_set = splitdata_train_test(data, 0.7)

  # generating the feature and targets for the training and test sets
  # i.e. train_features, train_targets, test_features, test_targets
  features_training, targets_training = generate_features_targets(training_set)
  features_testing, targets_testing = generate_features_targets(testing_set)

  # instantiating a decision tree classifier
  dtc = DecisionTreeClassifier()

  # training the classifier with
  # the train_features and train_targets
  dtc.fit(features_training, targets_training)

  # getting the predictions for the test_features
  predictions = dtc.predict(features_testing)

  # returning the predictions and the test_targets
  return predictions, targets_testing
```

Figure 34: Subroutine to Train the Decision Tree.

Continuing from the split function, the decision tree classifier is ready to be trained. For this, the function `dtc_predict_actual(data)` shown in figure34 is called.The purpose of this function is

4.3.7 Accuracy and model score

The accuracy for decision tree classifier is simpler to calculate compared to regression classifiers. The accuracy measure is also known as the model score. The accuracy is calculated as:

$$A = \frac{\sum_{i=1}^{N} P_i = A_i}{N}$$

where, A = accuracy,
P_i = predicted value,
A_i = actual value,
N = size of training set

`sklearn` has methods to get the model score. Most models will have a score method which in the case of the decision tree classifier uses the above formula.
The `cross_val_score` function in the `model_selection` module can be used to get k cross validated scores. The subroutine to calculate the accuracy is shown below in figure 35

```
import numpy as np
from matplotlib import pyplot as plt,rc
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict
from sklearn.tree import DecisionTreeClassifier
import itertools
from support_functions import plot_confusion_matrix,
 generate_features_targets
rc('font',**{'family':'serif','serif':['Helvetica']})
rc('text', usetex=True)

def calculate_accuracy(predicted, actual):
    correct = 0
    # iterating over the two lists simultaneously
    for p, a in zip(predicted, actual):
        if p == a:
            correct += 1
    accuracy = correct / len(actual)
    return accuracy
```

Figure 35: Subroutine to calculate the classification accuracy.

In addition to an overall accuracy score, we'd also like to know where our model is going wrong. For example, were the incorrectly classified mergers misclassified as spirals or ellipticals? To answer this type of question we use a confusion matrix. The subroutine to generate the matrix is shown below in figure 36

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Reds):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, "{}".format(cm[i, j]),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True Class')
    plt.xlabel('Predicted Class')
```

Figure 36: Subroutine to generate the confusion matrix.

The method invocation is done as:

```python
if __name__ == "__main__":
    data = np.load('galaxy_catalogue.npy')

    # splitting the data
    features, targets = generate_features_targets(data)

    # training the model to get predicted and actual classes
    dtc = DecisionTreeClassifier()
    predicted = cross_val_predict(dtc, features, targets, cv=10)

    # calculating the model score using your function
    model_score = calculate_accuracy(predicted, targets)
    print("Our accuracy score:", model_score)

    # calculating the models confusion matrix
    #  using sklearns confusion_matrix function
    class_labels = list(set(targets))
    model_cm = confusion_matrix(
        y_true=targets,
        y_pred=predicted,
        labels=class_labels)

    # Plot the confusion matrix
    #  using the provided functions.
    plt.figure()
    plot_confusion_matrix(
        model_cm,
        classes=class_labels,
        normalize=False)
    plt.savefig('confusion_matrix.png')
    plt.show()
```

Figure 37: Invocation of subroutines to train and assess the classification model.

The matrix is explained in the 'Results with Discussion) section in 5.2.1

4.3.8 Ensemble Learning

So far, a single decision tree has been used. However, there can be further improvements in the classification accuracy by employing an ensemble of decision trees also known as a random forest. A random forest averages different independently trained decision trees built from different subsets of the training set (Rebala, Ravi, and Churiwala 2019: 89).

When making a prediction, every tree in the forest gives its own prediction and the most common classification is taken as the overall forest prediction (in regression the mean prediction is used).

Random forests reduce overfitting tendency with decision trees.

- Training data is spread across decision trees. The subsets are created by taking random samples with replacement. This means that a given data point can be used in several subsets. This is different from the subsets used in cross-validation where each data point belongs to one subset.
- Individual trees are trained with different subsets of features. So in the galaxy's morphological classification, one tree might be trained using eccentricity and another using concentration and the /nth4 adaptive moment. By employing different combinations of input features expert trees are created that can better identify classes by a given feature.

The `sklearn` random forest only uses the first form of sampling. The `rf_predict_actual` subroutine returns the predicted and actual classes for the galaxies using random forest with 10-fold cross-validation.

You should use the `RandomForestClassifier` class from the `sklearn.ensemble` module. It can be instantiated as:

```
rfc = RandomForestClassifier(n_estimators=
↪ n_estimators)
```

`n_estimators` is the the number of decision trees in the forest.

`rf_predict_actual` takes two arguments: the data used throughout the problem and the number of estimators (`n_estimators`) to be used in the random forest. The subroutine returns two NumPy arrays containing the predicted and actual classes respectively.The `generate_features_targets` subroutine is reused. This approach gets the prediction for every galaxy in the data set through cross-validation. It also means that we don't need to manage the training and test sets.

```python
def rf_predict_actual(data, n_estimators):
    # generate the features and targets
    features, targets = generate_features_targets(data)

    # instantiate a random forest
    # classifier using n estimators
    rfc = RandomForestClassifier(n_estimators=n_estimators)

    # get predictions using 10-fold
    # cross validation with cross_val_predict
    predicted = cross_val_predict(rfc, features, targets, cv=10)

    # return the predictions and their actual classes
    return predicted, data['class']
```

Figure 38: Subrouting to perform 10-Fold cross validation with random forest.

```python
if __name__ == "__main__":
    data = np.load('galaxy_catalogue.npy')

    # get the predicted and actual classes
    number_estimators = 90 # Number of trees
    predicted, actual = rf_predict_actual(data, number_estimators)

    # calculate the model score using your function
    accuracy = calculate_accuracy(predicted, actual)
    print("Accuracy score:", accuracy)

    # calculate the models confusion matrix
    #  using sklearns confusion_matrix function
    class_labels = list(set(actual))
    model_cm = confusion_matrix(
        y_true=actual, y_pred=predicted, labels=class_labels)

    # plot the confusion matrix using the
    #  provided functions.
    plt.figure()
    plot_confusion_matrix(model_cm, classes=class_labels, normalize=False)
    plt.show()
```

Figure 39: Invocation of subroutine to perform 10-Fold cross validation with Random Forest.

5 Findings & Discussion

"In God we trust, all others bring data."

Dr. William E. Demming(1900-1993)
Columbia University

5.1 Red-shift Regression

5.1.1 Phase I : Accuracy Assessment with Median Difference

The first phase initialised the learning on the regression problem where redshifts of galaxies were predicted using flux magnitudes across 5 colour indices. This was the foundational phase which constituted feature and target extraction from the dataset. The decision tree algorithm from the scikit-learn package's 'DecisionTreeRegressor' to learn from the prepared dataset. The prediction was performed for input features from the same dataset. In order to assess the model's accuracy median of residuals i.e. the difference of actual and predicted value was used. Though the median residual strategy inherently did not have any problems with the precision of accuracy measurement, the problem was with the learning strategy. The 0 residual means that every prediction was successful. However, It was also unrealistic. This so because the training and testing set was a common one and the model was optimised to get the best results in the training set. The next phase of hold-out validation addressed these issues and a more realistic accuracy was obtained.

5.1.2 Phase II: Realistic accuracy with hold-out validation

Extending from the foundational first phase, to get a more realistic assessment of the decision tree's performance hold-out validation was introduced in this phase. Contrary to the common training and testing set used in the first phase, the dataset in this phase was split into two - the training subset and the testing or the validation subset. The regressor learnt from the training set which contained the `train features` and the corresponding `target feature`. The prediction was done using the `test features` of the training set. The median residual difference was calculated based on the difference of the model's target features produced during prediction and the actual target features.

The median difference of ≈ 0.02 was obtained. This means 50% of the dataset have < 0.02 residual error in the prediction which is a significantly low number. This shows a high accuracy of the decision tree regressor. The reason for choosing the median of differences as the accuracy metric is that it gives a fair representation of the errors. Since the distribution residuals as seen below in figure **??**, is skewed, the median is a good approximate to take rather than another measure for example the mean.

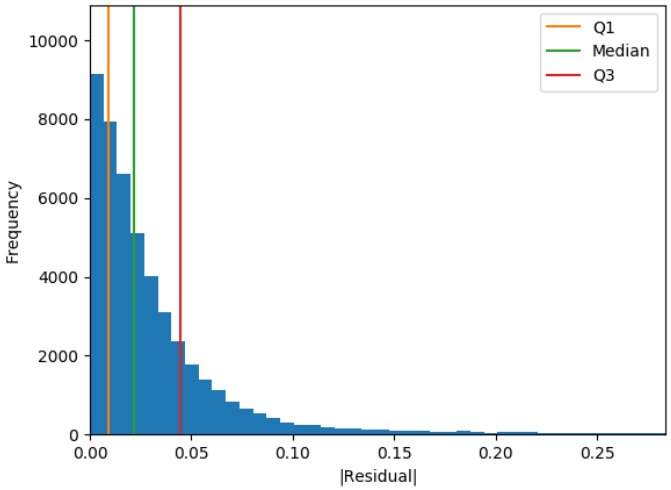


Figure 40: Residual distribution with quartiles.

5.1.3 Phase III: Improving the accuracy with K-Fold Cross Validation

The final phase attempted to improve the accuracy of the classifier. The foremost approach to improving the accuracy of any machine learning model would be to increase the size of the training set. However, in this case, augmenting training size seems to have little impact on accuracy. As shown in the graph below in figure 41, the median difference has difficulty surpassing the previously found accuracy of 0.02, even when the training size grows at 10 times the rate. Hence, for the training size of 25000, the accuracy limit has been reached.

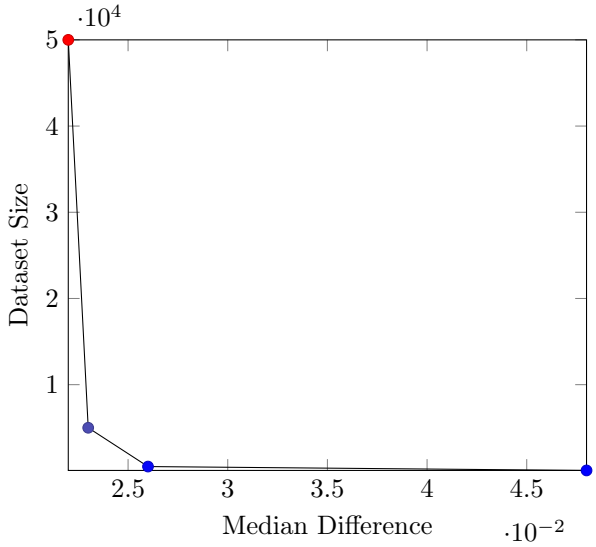


Figure 41: Rate of median difference change over the increasing dataset.

In order to further improve the accuracy, k-fold cross-validation method was used. This not only improves the accuracy but also mitigates the overfitting problem of the decision tree, simultaneously. The following graph depicts the overfitting problem in the previous phase. As visualised below in figure 42, as the tree depth increases, the model from the training set seem to have decreasing median residual or increasing performance. At a depth of 27, the errors seem to reduce to 0. On the other hand, the accuracy measure of the predictions on the test set though initially improves but eventually stars to overfit. In trying

to account for the outliers it loses its general predictive accuracy.

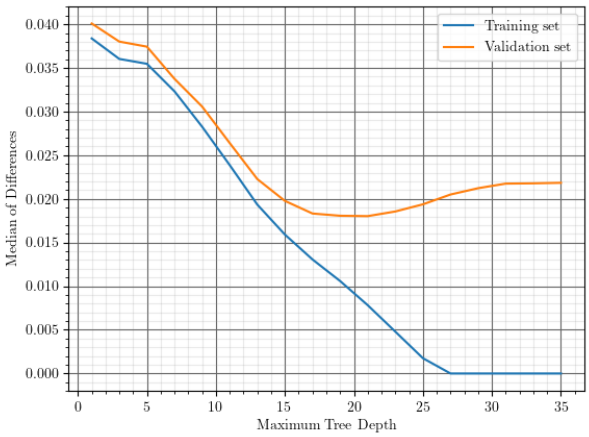


Figure 42: Overfitting problem in decision tree visualised.

The chart below shown in 43visualises the accuracy of the decision tree regression classifier after the k-fold cross validation. The dense points converge in a constant rate with occasional outliers.

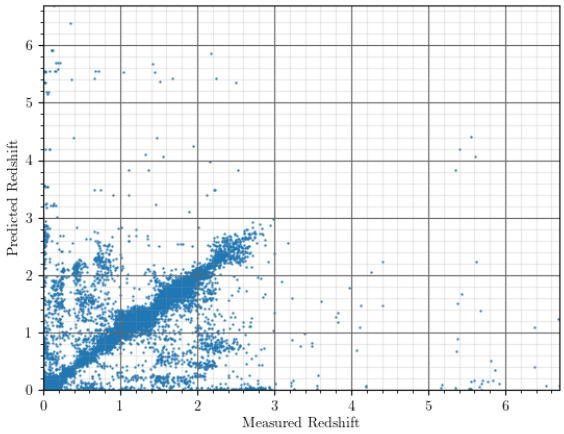


Figure 43: Visualisation of accuracy after k-fold validation.

5.1.4 Quasars vs Galaxies

The QSOs have greater median residual (≈ 0.074) than the galaxies (≈ 0.016). Possible reasons can be:

- Fewer number of QSOs in the dataset(8525) than galaxies(41,475).
- At SDSS flux may be feeble for SDSS at ≈ 0.4. This can create a measurement bias.

The median residual on a random sample of galaxies was _diff of ≈ 0.018 which is slightly higher than the complete subset, but not sufficient to accommodate the gap between the two populations.

The figure below shows the normalised distribution function of the two populations.

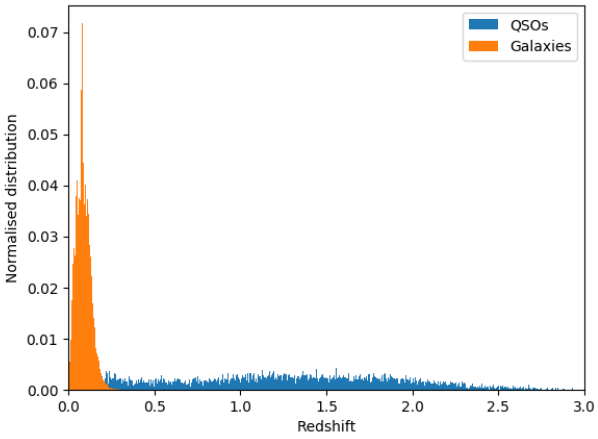


Figure 44: Distribution of redshifts of galaxies and quasars

The majority of galaxies accumulate at 0.10 to form a peak while the quasars are spread out and evenly distributed out up to redshift ≈ 2.5. This could result in measurement bias. In the case of the galaxies, the decision tree is trained to approximate around 0.4. As such the predictions from this model will not be larger than the maximum target value. So the maximum difference (or residual) for each galaxy in this set will be a lot smaller than the maximum residual for the QSOs.

A clearer view of this by looking at the predicted redshifts vs actual redshifts in the following plot.

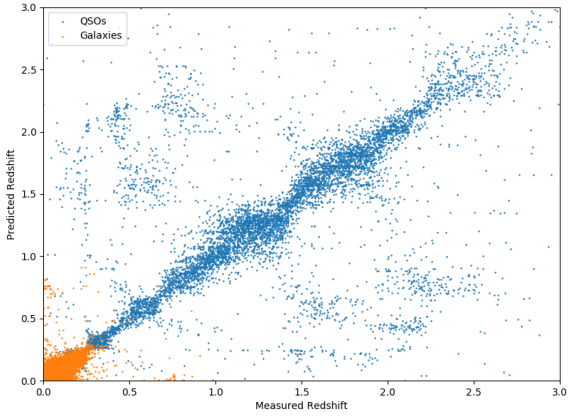


Figure 45: Predicted vs measured redshifts of galaxies and quasars.

5.2 Morphology Classification

5.2.1 Accuracy Assessment with Confusion Matrix

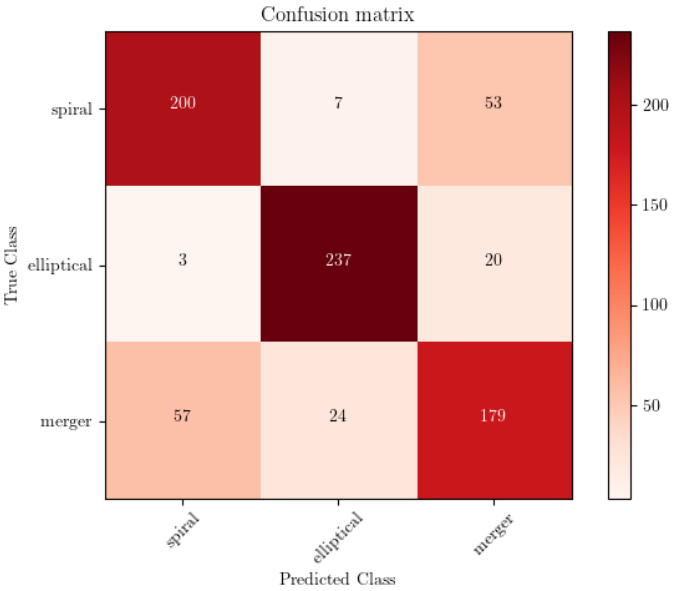


Figure 46: Confusion matrix for elliptical, mergers, spiral galaxies.

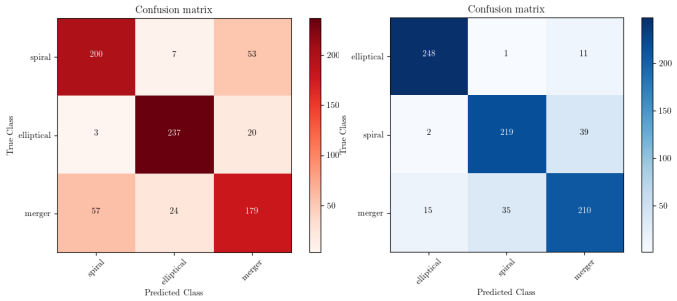
The x-axis represents the predicted classes and the y-axis represents the correct classes. The value in each cell is the number of examples with those predicted and actual classes. Correctly classified objects are along the diagonal of the matrix. So of the 260 actual spirals (correct class) in the data set, 200 are correctly predicted as spirals, 3 are incorrectly predicted as ellipticals and 57 are incorrectly predicted as mergers.The sum along each row or column can be used to get the totals of true and predicted classes. So for example, by summing each of the rows it can be confirmed there are 260 mergers, 260 spirals and 260 ellipticals in the data set.

5.2.2 Random Forest vs Decision Tree

Did the random forest improve the accuracy of the model? The answer is yes – we see a substantial increase in accuracy. When we look at the 10-fold cross validation results, we see that the random forest systematically out-performs a single decision tree:

Metric	Random Forest	Decision Tree
Median score (Mdn)	0.865	0.775
Mean score (μ)	0.867	0.792
Standard deviation of scores (σ)	0.036	0.035

Table 3: Performance of Random Forest vs Decision Tree



(a) Confusion Matrix of Decision Tree. **(b)** Confusion Matrix of Random Forest

Figure 47: A comparison between prediction accuracy for decision tree and ensemble method used in morphologic classification of galaxies.

The random forest is around 6-7% more accurate than a standard decision tree. The figure above depicts a side-by-side comparison of the confusion matrices from the galaxy catalogues. The second showing the results for the random forest classifier and the first for the decision tree classifier. There are improvements across the board with the biggest improvement (percentage) observed between ellipticals and spirals.

6 Conclusion

> "The heavens declare the glory of God, and the sky above proclaims his handiwork.
>
> Psalm 19:1

6.1 Lessons Learnt

- In the regression problem, based on the flux magnitudes of 5 colour indices, decision tree models were used to predict its redshift. To assess the accuracy, the median of the residuals was used as an accuracy measure.
- Some non-machine learning approaches were discussed such as model fitting and comparison with spectral emission lines but their limitations demanded the employment of machine learning.
- A basic hold-out validation was performed and which was unrealistic because it was optimised to work for the training data. The problem of overfitting was also explored. To resolve the issue, K-fold cross-validation was introduced.
- The decision trees were prone to overfitting the model and limiting the maximum depth of the tree, could prevent it. By comparing the accuracy of the model on the training set with that of the test set for different tree depths it was found out that a maximum tree depth of 19 was optimal for the model.
- k-fold cross-validation which is also shipped as Python libraries were implemented. A custom method was developed to understand the inner-workings. K-fold cross-validation mitigated the risk that the training set had a class imbalance. In this case, few quasars

in the mix of regular galaxy dataset were classified without disrupting the model's efficacy. K-fold was pivotal in mitigating measurement bias resulting from the difference in the range of redshifts in each type of population.

- It was shown that the decision tree can work for both the regression and the classification problem. The working mechanism was pretty much the same in both cases. The former returned a real-valued number whereas the later was a discrete value, in this case, the morphology of galaxy.
- In the classification part, decision tree classifiers used multitudes of features of galaxies to identify a galaxies type by a selection of derived features (e.g. eccentricity and concentration).
- It was found that assessing the model's performance is a lot simpler with classification than it is with regression. Confusion matrices can be a useful tool to help understand where the model is over and under performing with respect to each class. I started out with the goal of using decision trees to classify galaxies as one of three types. I was able to achieve an accuracy of around 80% using the decision trees which is very good when one considers the statistical accuracy of a random selection is 33% and the naive approach using only pixel values yielded close to 64%. The study was able to improve on the accuracy of the decision tree classifier by using a selection of them in ensemble learning with a random forest.

6.2 Limitation of Study

- Colour has been the basis for prediction of morphology and redshifts for galaxies in this study. In terms of morphology, a limitation in my study is it does not take into account the galaxy evolution story. The colour of a galaxy is mainly determined by its stellar content, gas and dust. While the morphology of a galaxy evinces at the dynamical history and the two do not necessarily share common timescale. To resolve the aforementioned issues, a large dataset of the early-type and late-type galaxy which is independent of colour needs to be taken into account and different classifiers with other training features must be selected and tested out.
- With regards to the redshift prediction, the model does not consider the photometric uncertainty. And sampling can introduce systematic errors which can bias variability metrics. The uncertainty can be considered in the next study or efficiency-purity of systematic errors can be shown explicitly.

6.3 Future Work

- Datasets including large sets of early-type and late-type galaxies can be taken into consideration to classify galaxies based on other features independent of colour.
- Systematic bias can be eliminated by taking into account primary measurement residuals into the learning algorithm.
- A richer web-based user interface with robust exception handling in place, can be developed to steer the prediction from feature extraction to class generation.
- The dissertation can be further edited and made publication-ready. Astrophysical and machine learning journals can be approached for publishing based on this study.

7 Acknowledgment

I am grateful to my teacher Achyut Timisina, the academic director of computing at Softwarica. His piercing pedagogy and facilitation were formative to my growth. But I am touched by his cordial demeanor. I am also thankful to my supervisor Manoj Shrestha.

I am indebted to Dr Tara Murphy and Dr Simon Murphy, professors of Computational Astronomy at the University of Sydney whose MOOC on 'Data-driven Astronomy' laid in me the foundations of applied machine learning to solve astronomical problems.

Heartfelt gratitude is due to Dr William Lane Craig, the research professor of Philosophy at Biola University, who galvanised in me the existential questions of ultimate origin, meaning, morality, and destiny; and taught me the life of the mind.

I cannot thank enough my lord and saviour Jesus of Nazareth. He is the sole embodiment of the truth, the way, and the life, who strengthens me.

References

Allwein, Erin L., Robert E. Schapire, and Yoram Singer (September 2001). "Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers". In: *The Journal of Machine Learning Research* 1, pp. 113–141.

Ball, N. M., J. Loveday, M. Fukugita, O. Nakamura, S. Okamura, J. Brinkmann, and R. J. Brunner (March 2004). "Galaxy types in the Sloan Digital Sky Survey using supervised artificial neural networks". In: *Monthly Notices of the Royal Astronomical Society* 348.3, pp. 1038–1046.

Banfield, Robert E., Lawrence O. Hall, Kevin W. Bowyer, and W.P. Kegelmeyer (January 2007). "A Comparison of Decision Tree Ensemble Creation Techniques". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.1, pp. 173–180.

Barchi, P. H., R. R. de Carvalho, R. R. Rosa, R. A. Sautter, M. Soares-Santos, B. A. D. Marques, E. Clua, T. S. Gonçalves, C. de Sá-Freitas, and T. C. Moura (January 1, 2020). "Machine and Deep Learning applied to galaxy morphology - A comparative study". In: *Astronomy and Computing* 30.

Barragan-Jason, G., M. Cauchoix, and E. J. Barbeau (August 1, 2015). "The neural speed of familiar face recognition". In: *Neuropsychologia* 75, pp. 390–401.

Bazell, D. and David W. Aha (February 2001). "Ensembles of Classifiers for Morphological Galaxy Classification". In: *The Astrophysical Journal* 548.1, pp. 219–223.

Benediktsson, Jón Atli, Josef Kittler, and Fabio Roli, eds. (2009). *Multiple Classifier Systems: 8th International Workshop, MCS 2009*. Vol. 5519. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg.

Breiman, Leo (1996). *Bias, variance, and arcing classifiers*. Technical Report 460. Statistics Department, University of California, Berkeley.

Buta, Ronald J. (2013). "Galaxy Morphology". In: *Planets, Stars and Stellar Systems: Volume 6: Extragalactic Astronomy and Cosmology*. Ed. by Terry D. Oswalt and William C. Keel. Dordrecht: Springer Netherlands, pp. 1–89.

Caharel, Stephanie, Meike Ramon, and Bruno Rossion (August 5, 2013). "Face Familiarity Decisions Take 200 msec in the Human Brain: Electrophysiological Evidence from a Go/No-go Speeded Task". In: *Journal of Cognitive Neuroscience* 26.1, pp. 81–95.

Cooper, Leon N and Michael P. Perrone (1993). "When networks disagree: Ensemble methods for hybrid neural networks". In: *Artificial Networks for Speech & Vision*. Ed. by R.J. Mammone. London: Chapman & Hall, pp. 126–142.

Craig, William Lane (December 1, 1999). "The Ultimate Question of Origins: God and the Beginning of the Universe". In: *Astrophysics and Space Science* 269, pp. 721–738.

Davies, Paul (2007). *God & The New Physics*. New York: Simon and Schuster Paperbacks. 255 pp.

De Condorcet, Nicolas (2014). *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. Cambridge: Cambridge University Press.

Dietterich, Thomas G. (August 1, 2000). "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization". In: *Machine Learning* 40.2, pp. 139–157.

Domingos, Pedro (2019). *What Is Machine Learning?* Intel. URL: `https://www.intel.com/content/www/in/en/analytics/ai-luminary-pedro-domingos-video.html` [13 December 2019].

Drucker, Harris, Corinna Cortes, L. D. Jackel, Yann LeCun, and Vladimir Vapnik (November 1994). "Boosting and Other Ensemble Methods". In: *Neural Computation* 6.6, pp. 1289–1301.

European Space Agency (2019). *ESA Science & Technology - NGC 5257 and NGC 5258*. European Space Agency. URL: `https://sci.esa.int/web/hubble/-/42656-ngc-5257-and-ngc-5258` [9 August 2020].

Evan Gough and National Astronomical Observatory of Japan (NAOJ) (June 19, 2019). *Artist's impression of the merging galaxies B14-65666 located 13 billion light-years away*. Universe Today. URL: `https://www.universetoday.com/142578/the-earliest-example-of-merging-galaxies-ever-found/` [30 September 2019].

Filippenko, Alex (2007). *The Great Courses Guidebook - Understanding the Universe: An Introduction to Astronomy*. Second Edition. Chantilly, Virginia: The Great Courses.

Filippi, E., M. Costa, and E. Pasero (1994). "Multi-layer perceptron ensembles for increased performance and fault-tolerance in pattern recognition tasks". In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*. 1994 IEEE International Conference on Neural Networks (ICNN'94). Vol. 5. Orlando, FL, USA: IEEE, pp. 2901–2906.

Guttag, John (2016). *Introduction to Computation and Programming using Python: With Application to Understanding Data*. 2nd. Cambridge, Massachusetts: The MIT Press.

Guyer, Paul, ed. (January 31, 1992). *The Cambridge Companion to Kant*. 1st ed. Cambridge University Press.

Haindl, Michal, Josef Kittler, and Fabio Roli, eds. (2007). *Multiple Classifier Systems: 7th International Workshop, MCS 2007, Prague, Czech Republic, May 23-25, 2007, Proceedings*. Image Processing, Computer Vision, Pattern Recognition, and Graphics. Berlin Heidelberg: Springer-Verlag.

Ho, Tin Kam (May 2002). "Multiple Classifier Combination: Lessons and Next Steps". In: Bunke, H and A Kandel. *Series in Machine Perception and Artificial Intelligence*. Vol. 47. World Scientific, pp. 171–198.

Hubble, Edwin (March 15, 1929). "A relation between distance and radial velocity among extra-galactic nebulae". In: *Proceedings of the National Academy of Sciences* 15.3, pp. 168–173.

Hubble, Edwin, Robert P. Kirshner, and Sean M. Carroll (2013). *The Realm of the Nebulae*. Mrs. Hepsa Ely Silliman Memorial Lectures. New Haven: Yale University Press.

IEEE Computer Society Digital Liibrary (1999). *IEEE Computer Society Digital Liibrary*. The Sloan Digital Sky Survey. URL: `https://csdl-images.computer.org/mags/cs/1999/02/figures/c20541.gif` [10 October 2019].

Ivezić, Željko, Andrew Connolly, Jacob T. VanderPlas, and Alexander Gray, eds. (2014). *Statistics, Data min-*

ing, and Machine Learning in Astronomy: A Practical Python Guide for The Analysis of Survey Data. Princeton Series in Modern Observational Astronomy. OCLC: ocn841893567. Princeton, N.J: Princeton University Press.

Jet Propulsion Laboratory, California Institute of Technology (2019). *Home | NASA/IPAC Extragalactic Database*. NED. URL: https://ned.ipac.caltech.edu/ [29 July 2019].

Kant, Immanuel (2012). *Kant: Natural Science*. Ed. by Eric Watkins. Cambridge: Cambridge University Press.

Kittler, J., M. Hatef, R.P.W. Duin, and J. Matas (March 1998). "On combining classifiers". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.3, pp. 226–239.

Kleinberg, E.M. (May 2000). "On the algorithmic implementation of stochastic discrimination". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.5, pp. 473–490.

Kuncheva, Ludmila Ilieva (2014). *Combining pattern classifiers: methods and algorithms*. 2nd ed. Hoboken, NJ: Wiley. 357 pp.

Lahav, O., A. Naim, R. J. Buta, H. G. Corwin, G. de Vaucouleurs, A. Dressler, J. P. Huchra, S. van den Bergh, S. Raychaudhury, L. Sodre, and M. C. Storrie-Lombardi (February 10, 1995). "Galaxies, Human Eyes, and Artificial Neural Networks". In: *Science* 267.5199, pp. 859–862.

Lam, Louisa and Ching Y. Suen (September 1995). "Optimal combinations of pattern classifiers". In: *Pattern Recognition Letters* 16.9, pp. 945–954.

Liitiäinen, Elia, Michel Verleysen, Francesco Corona, and Amaury Lendasse (October 2009). "Residual variance estimation in machine learning". In: *Neurocomputing* 72.16, pp. 3692–3703.

Lintott, Chris J., Kevin Schawinski, Anže Slosar, Kate Land, Steven Bamford, Daniel Thomas, M. Jordan Raddick, Robert C. Nichol, Alex Szalay, Dan Andreescu, Phil Murray, and Jan Vandenberg (September 21, 2008). "Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey". In: *Monthly Notices of the Royal Astronomical Society* 389.3, pp. 1179–1189.

Lintott, Chris, Kevin Schawinski, Steven Bamford, Anže Slosar, Kate Land, Daniel Thomas, Edd Edmondson, Karen Masters, Robert C. Nichol, M. Jordan Raddick, Alex Szalay, Dan Andreescu, Phil Murray, and Jan Vandenberg (January 1, 2011). "Galaxy Zoo 1: data release of morphological classifications for nearly 900 000 galaxies". In: *Monthly Notices of the Royal Astronomical Society* 410.1, pp. 166–178.

McCarthy, John and Edward A. Feigenbaum (September 15, 1990). "In Memoriam: Arthur Samuel: Pioneer in Machine Learning". In: *AI Magazine* 11.3, pp. 10–10.

Mitchell, Tom M. (1997). *Machine Learning*. New York: McGraw-Hill.

– (November 1, 1999). "Machine learning and data mining". In: *Communications of the ACM* 42.11, pp. 30–36.

NASA (2016). *Active Galaxies and Quasars - Introduction*. URL: https://imagine.gsfc.nasa.gov/science/objects/active_galaxies1.html [18 August 2019].

NASA/Goddard Space Flight Center Conceptual Image Lab (2008). *Active Galactic Nucleus*. URL: http://svs.gsfc.nasa.gov/20135.

NASA/JPL (2019). *The Milky Way Galaxy*. NASA-Solar System Exploration. URL: https://solarsystem.nasa.gov/resources/285/the-milky-way-galaxy/ [9 September 2019].

NASA/JPL Caltech (2017). *The Pinwheel Galaxy*. NASA. URL: http://www.nasa.gov/multimedia/imagegallery/image_feature_2265.html [1 August 2019].

NASA and ESA/Hubble (2016). *Hubble Views a Galactic Mega-merger*. URL: http://www.nasa.gov/image-feature/goddard/2016/hubble-views-a-galactic-mega-merger [30 October 2019].

NASA and Space Telescope Science Institute (2004). *HubbleSite: Image - Hubble Ultra Deep Field*. URL: https://hubblesite.org/image/3886/category/58-hubble-ultra-deep-field [30 December 2019].

Oza, Nikunj C., Robi Polikar, Josef Kittler, and Fabio Roli, eds. (2005). *Multiple Classifier Systems: 6th International Workshop, MCS 2005*. Image Processing, Computer Vision, Pattern Recognition, and Graphics. Berlin Heidelberg: Springer-Verlag.

Raddick, M. Jordan, Georgia Bracey, Pamela L. Gay, Chris J. Lintott, Carie Cardamone, Phil Murray, Kevin Schawinski, Alexander S. Szalay, and Jan Vandenberg (March 27, 2013). "Galaxy Zoo: Motivations of Citizen Scientists". In:

Raddick, M. Jordan, Georgia Bracey, Pamela L. Gay, Chris J. Lintott, Phil Murray, Kevin Schawinski, Alexander S. Szalay, and Jan Vandenberg (December 2010). "Galaxy Zoo: Exploring the Motivations of Citizen Science Volunteers". In: *Astronomy Education Review* 9.1.

Rebala, Gopinath, Ajay Ravi, and Sanjay Churiwala (2019). *An Introduction to Machine Learning*. Switzerland: Springer International Publishing.

Roli, Fabio, Josef Kittler, and Terry Windeatt, eds. (2004). *Multiple Classifier Systems: 5th International Workshop, MCS 2004*. Lecture Notes in Computer Science. Berlin Heidelberg: Springer-Verlag.

Ross, Hugh (2008). *Why The Universe Is The Way It Is?* Grand Rapids, Michigan: Baker Books. 240 pp.

Samuel, A. L. (July 1959). "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of Research and Development* 3.3, pp. 210–229.

Sandage, Allan (1984). *The Hubble Atlas of Galaxies*. Washington D.C.

Schlesinger Library (May 27, 2016). *Portrait of Annie Jump Cannon, ca. 1895-1897*. URL: https://www.flickr.com/photos/schlesinger_library/27219643971/.

Schneider, Jeff (2019). *Cross Validation*. Jeff Schneider| cs.cmu.edu/~schneider. URL: https://www.cs.cmu.edu/~schneide/tut5/node42.html [10 June 2019].

Shaw, Jonathan (December 6, 2018). *Artificial Intelligence and Ethics*. Harvard Magazine. URL: https://harvardmagazine.com/2019/01/artificial-intelligence-limitations [19 July 2019].

Sloan Digital Sky Survey (2014). *Sloan Digital Sky Survey*. URL: http://classic.sdss.org/ [15 November 2019].

– (2019). *Measures of Flux and Magnitude | SDSS*. URL: https://www.sdss.org/dr12/algorithms/magnitudes/#mag_petro [19 October 2019].

Sohn, S.Y. and H.W. Shin (January 2007). "Experimental study for the comparison of classifier combination methods". In: *Pattern Recognition* 40.1, pp. 33–40.

Space Telescope Science Institute (2007). *HubbleSite: Image - ESO 325-G004: Detailing the Big Picture*. STScI. URL: https://hubblesite.org/image/2057/news/55-globular-clusters [9 August 2019].

Stanford University (2019). *Professor Arthur Samuel | Stanford Computer Science*. URL: https://cs.stanford.edu/memoriam/professor-arthur-samuel [8 September 2019].

Stoughton, Chris, Robert H. Lupton, Mariangela Bernardi, Michael R. Blanton, Scott Burles, Francisco J. Castander, ..., and Idit Zehavi (January 2002). "Sloan Digital Sky Survey: Early Data Release". In: *The Astronomical Journal* 123.1, pp. 485–548.

Strauss, Michael A., David H. Weinberg, Robert H. Lupton, Vijay K. Narayanan, James Annis, Mariangela Bernardi, ..., and Idit Zehavi (September 1, 2002). "Spectroscopic Target Selection in the Sloan Digital Sky Survey: The Main Galaxy Sample". In: *The Astronomical Journal* 124, pp. 1810–1824.

Sung-Bae Cho and J.H. Kim (March 1995). "Multiple network fusion using fuzzy logic". In: *IEEE Transactions on Neural Networks* 6.2, pp. 497–501.

Thagard, Paul (1990). "Philosophy and Machine Learning". In: *Canadian Journal of Philosophy* 20.2, pp. 261–276.

The Institute for Ethical AI & Machine Learning (2019). *The Machine Learning Principles*. URL: https://ethical.institute [19 September 2019].

Thonnat, Monique (1989). "Toward an Automatic Classification of Galaxies". In: *The World of Galaxies*. Ed. by Harold G. Corwin and Lucette Bottinelli. New York, NY: Springer US, pp. 53–74.

Vaucouleurs, Gérard-Henri de, ed. (1991). *Third reference catalogue of bright galaxies*. New York, NY: Springer.

Wakabayashi, Daisuke (March 19, 2018). "Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam". In: *The New York Times*. URL: https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html [19 December 2019].

Way, Michael J., Jeffrey D. Scargle, Kamal M. Ali, and Ashok N. Srivastava (March 29, 2012). *Advances in Machine Learning and Data Mining for Astronomy*. CRC Press.

Wolbach Library of the Harvard-Smithsonian Center for Astrophysics (2019). *Astronomy and Spectroscopy*. John G. Wolbach Library. URL: https://library.cfa.harvard.edu/cannon-spectroscopy [11 November 2019].

Wu, Xue-Bing, Feige Wang, Xiaohui Fan, Weimin Yi, Wenwen Zuo, Fuyan Bian, Linhua Jiang, Ian D. McGreer, Ran Wang, Jinyi Yang, Qian Yang, David Thompson, and Yuri Beletsky (February 2015). "An ultraluminous quasar with a twelve-billion-solar-mass black hole at redshift 6.30". In: *Nature* 518.7540, pp. 512–515.

York, Donald G., J. Adelman, John E. Anderson Jr., Scott F. Anderson, James Annis, Neta A. Bahcall, ..., and Naoki Yasuda (September 2000). "The Sloan Digital Sky Survey: Technical Summary". In: *The Astronomical Journal* 120.3, pp. 1579–1587.

8 Appendix

8.1 Word Count

$\tilde{1}$3000

8.2 Dissertation Structure

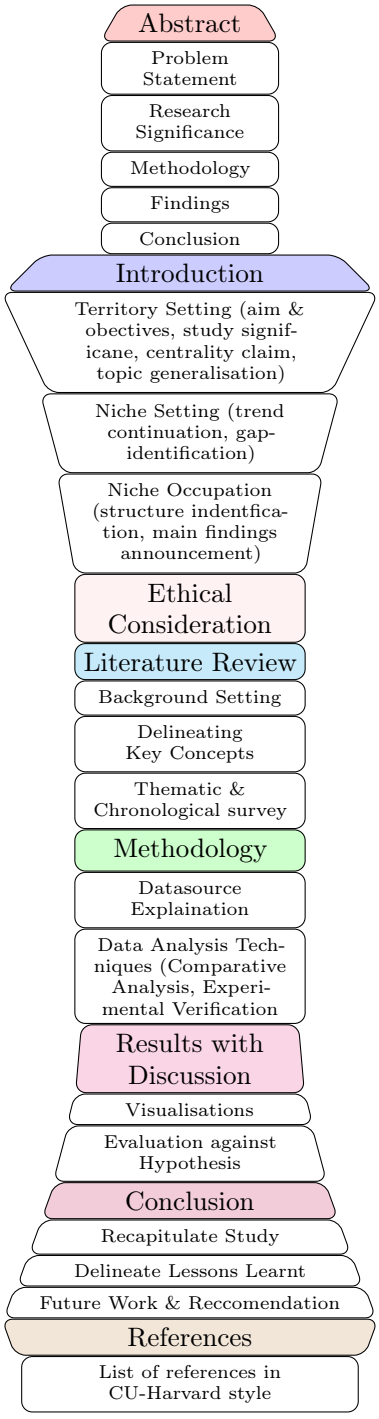


Figure 48: Custom-designed ordered Structure for my Dissertation

8.3 Visual Classification Scheme

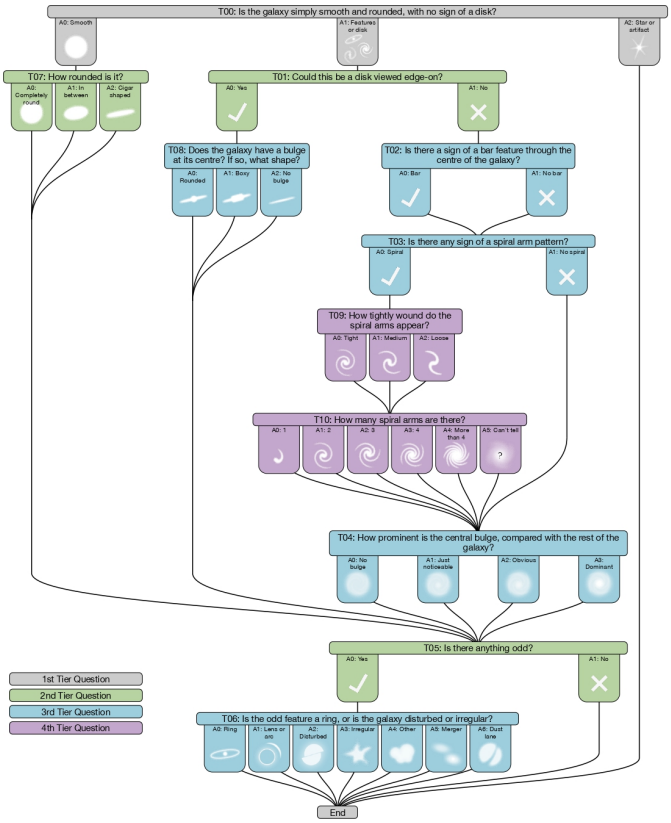


Figure 49: Galaxy Zoo Decision Tree for Visual Classification

8.4 Code Snippets

8.5 GUI Prototype

The system developed does not have a graphical user interface. It uses command-line user interface. I have however developed a prototype for the GUI using the Django web-framework. The file structure is shown in figure 50 The CLI routines are integrated into the view controllers in Django as shown in figure 53 and 54. The corresponding source code of the rendered page is shown in figure 51 and 52. The rendered page itself is shown in figure 55

Figure 50: File structure for the GUI prototype developed in Django.

```
{% extends 'base.htm' %}
{% block content %}

{% if messages %}
{% for message in messages %}
<div class=
"alert alert-dark alert-{{ message.tags }} alert-dismissible fade
show"
 role="alert">
    {{ message }}
    <button type="button" class="close" data-dismiss="alert"
aria-label="Close">
        <span aria-hidden="true">&times;</span>
    </button>
</div>
{% endfor %}
{% endif %}

<h2>Morphology Classification</h2>
<div class="parent_div">
    <div class="dc1">
        <form method="POST" enctype="multipart/form-data">
            {% csrf_token %}
            <div class="form-group">
                <label for="file">Load Galaxy Catalogue (NumPy)
</label>
                <input type="file" name="file"><br>
            </div>
            <div class="form-group">
                <label for="training_fraction">Training Fraction
</label>
                <input class="form-control" type="number" min=
".1" step="0.01" name='training_fraction' value=".7">
            </div>
            <button id="btn_load_dataset" class=
"badge badge-pill" type="submit">Load Dataset</button>
        </form>
        <button class="badge badge-pill" id=
"btn_extract_features">Extract Features</button>
        <div id="div_features" style='
max-width:50px;margin:auto;'>
        </div>
    </div>

    <div class="dc2">
        <h3>Dataset Dimensions</h3>
        <table class="table table-sm table-dark">
            <tr><th># of Galaxies in Dataset</th><td>{{size}}
</td></tr><tr><th>Training Fraction</th><td>
{{n_training_fraction}}</td></tr>
            <tr><th># of Galaxies in Training Set</th><td>
{{n_training_set}}</td></tr>
            <tr><th># of Galaxies in Testing Set</th><td>
{{n_testing_set}}</td></tr>
        </table>
    </div>
```

Figure 51: An example of front-end design with HTML and Django Template (Part 1 of 2).

```
    <div class="dc3">
          <h3>Sample Record</h3>
          <table class="table table-sm table-dark">
              <tr><th>u-g</th><td>{{u_g}}</td></tr>
              <tr><th>g-r</th><td>{{g_r}}</td></tr>
              <tr><th>r-i</th><td>{{r_i}}</td></tr>
              <tr><th>i-z</th><td>{{i_z}}</td></tr>
              <tr><th>ecc</th><td>{{ecc}}</td></tr>
              <tr><th>m4_u</th><td>{{ m4_u }}</td></tr>
              <tr><th>m4_g</th><td>{{ m4_g }}</td></tr>
              <tr><th>m4_r</th><td>{{ m4_r }}</td></tr>
              <tr><th>m4_i</th><td>{{ m4_i }}</td></tr>
              <tr><th>m4_z</th><td>{{ m4_z }}</td></tr>
              <tr><th>ecc</th><td>{{ ecc }}</td></tr>
              <tr><th>petroR50_u</th><td>{{ petroR50_u }}</td></tr>

              <tr><th>petroR50_r</th><td>{{ petroR50_r }}</td></tr>

              <tr><th>petroR50_z</th><td>{{ petroR50_z }}</td></tr>

              <tr><th>petroR90_u</th><td>{{ petroR90_u }}</td></tr>

              <tr><th>petroR90_r</th><td>{{ petroR90_r }}</td></tr>

              <tr><th>petroR90_z</th><td>{{ petroR90_z }}</td></tr>

          </table>
      </div>

      <div class="dc4">
          {% load static %}
          <h3> Galaxy Class</h3>
          {% if class == 'spiral' %}
          <img ALIGN="left" src="{% static 'images/spiral.png' %}"
/>
          {% elif class == 'elliptical' %}
          <img ALIGN="left" src=
"{% static 'images/elliptical.png' %}" />
          {% elif class == 'merger' %}
          <img ALIGN="left" src="{% static 'images/merger.png' %}"
/>
          {% else %}
          <div style='
border: 1px solid olive; width:180px; height:180px;'></div>
          {% endif %}
          <br>
          Type: <
style="float:right;" span class="badge badge-pill badge-warning">
{{class|upper}}
</span>
      </div>
</div>
```

Figure 52: An example of front-end design with HTML and Django Template (Part 2 of 2).

```
from django.shortcuts import render
from django.core.files.storage import FileSystemStorage
import os
from random import randint
from django.contrib import messages
import numpy as np
import math

def show_home(request):
    return render(request, 'home.htm')

def show_load_c(request):
    data, tf = load_dataset(request)
    return render(request, 'load_c.html'
, get_dataset_preview(data, tf))

def load_dataset(request):
    data = None
    tf = None
    valid_extensions = ['.npy']

    if request.method == 'POST':
        file = request.FILES['file']
        fs = FileSystemStorage()
        # fs.save(file.name, file)
        ext = os.path.splitext(file.name)[1]
        if not ext.lower() in valid_extensions:
            messages.info(request,
'Invalid File. Please upload a valid NumPY file.')
            return data, tf
        else:
            # loading the file
            data = np.load(file)
            tf = float(request.POST.get('training_fraction'))
            messages.info(request, 'Dataset: \"' +file.name + '\"
 loaded Sucessfully!')
    return data, tf

def get_dataset_preview(data, tf):
    model = {}
    if data is None and tf is None:
        return model

```

Figure 53: An example of view controller with routines integrated in Django (Part 1 of 2).

```

# setting the fraction of data which should be in the training se
t

    fraction_training = tf

    # spliting the data
    training, testing =
 splitdata_train_test(data, fraction_training)

    # printing the key values
    model = {
        'size': len(data),
        'n_training_fraction': fraction_training,
        'n_training_set': len(training),
        'n_testing_set': len(testing)
    }

    # print an example
    for name, value in zip(data.dtype.names, data[randint(0, len
(data))]):
        # Make model characters Template-Friendly
        name = name.replace('-', '_')
        model[name] = value
    return model


def splitdata_train_test(data, fraction_training):
    # randomizing data set order
    np.random.seed(0)
    np.random.shuffle(data)

    # find split point
    training_rows = math.floor(len(data) * fraction_training)
    training_set = data[:training_rows]
    testing_set = data[training_rows:len(data)]

    return (training_set, testing_set)
```

Figure 54: An example of view controller with routines integrated in Django. (Part 2 of 2)

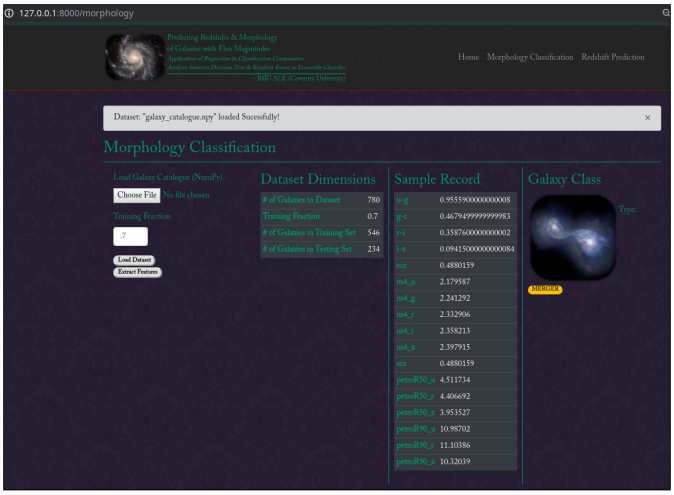


Figure 55: Sample page loaded with Django controllers as part of GUI prototype.

8.6 Redshift Regression

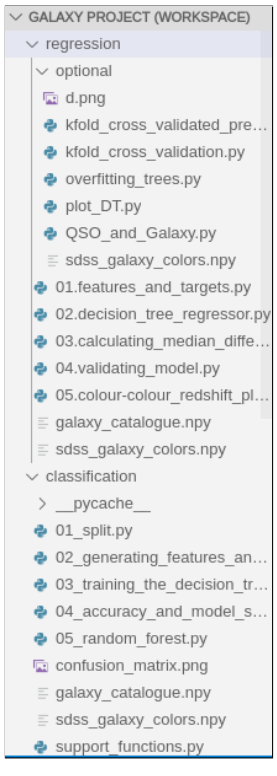


Figure 56: File structure of classification and regression on SDSS dataset.

```python
import numpy as np

def get_features_targets(data):
  #n rows, 4 columns
  features = np.zeros((data.shape[0], 4))
  features[:,0] = data['u'] - data['g']
  features[:,1] = data['g'] - data['r']
  features[:,2] = data['r'] - data['i']
  features[:,3] = data['i'] - data['z']

  targets = data['redshift']

  return (features, targets)

if __name__ == "__main__":
  # loading the data
  data = np.load('sdss_galaxy_colors.npy')

  features, targets = get_features_targets(data)

  # printing the shape of the returned arrays
  print(features[:2])
  print(targets[:2])
```

Figure 57: Features (u,g,r,i,z flux magnitudes) Extraction from SDSS dataset.

```python
import numpy as np
from sklearn.tree import DecisionTreeRegressor

def get_features_targets(data):
  #n lines, 4 columns
  features = np.zeros((data.shape[0], 4))
  features[:,0] = data['u'] - data['g']
  features[:,1] = data['g'] - data['r']
  features[:,2] = data['r'] - data['i']
  features[:,3] = data['i'] - data['z']
  targets = data['redshift']
  return (features, targets)

# loading the data and
data = np.load('sdss_galaxy_colors.npy')
# generating the features and targets
features, targets = get_features_targets(data)

# initializing model
dtr = DecisionTreeRegressor()

# training the model
dtr.fit(features, targets)

# making predictions using the same features
predictions = dtr.predict(features)

# printing the first 4 predicted redshifts
print(predictions[:4])
```

Figure 58: Training the decision tree regressor.

```python
import numpy as np

# calculating the median of the differences
# between our predicted and actual values
def median_diff(predicted, actual):
  diff = np.median(np.absolute(predicted - actual))
  return diff


if __name__ == "__main__":
  # load testing data
  targets = np.load('targets.npy')
  predictions = np.load('predictions.npy')

  # calling the function to measure
  #  the accuracy of the predictions
  diff = median_diff(predictions, targets)

  # print the median difference
  print("Median difference: {:0.3f}".format(diff))
```

Figure 59: Calculating median residuals.

```python
import numpy as np
from sklearn.tree import DecisionTreeRegressor
from matplotlib import pyplot as plt, rc

rc('font',**{'family':'serif','serif':['Helvetica']})
rc('text', usetex=True)

def get_features_targets(data):
  features = np.zeros((data.shape[0], 4)) #n lines, 4 columns
  features[:,0] = data['u'] - data['g']
  features[:,1] = data['g'] - data['r']
  features[:,2] = data['r'] - data['i']
  features[:,3] = data['i'] - data['z']
  targets = data['redshift']
  return (features, targets)

def median_diff(predicted, actual):
  diff = np.median(np.absolute(predicted - actual))
  plot_med_hst(predicted, actual)
  return diff

# Show a histogram of median distribution
def plot_med_hst(predicted, actual):
  d=[]
  for i in range(len(predicted)):
    d.append(abs(predicted[i]-actual[i]))
  plt.hist(d, bins='auto')
  plt.xlabel('Median Residual')
  plt.ylabel('Frequency')
  plt.xlim(0, .5)
  plt.grid(b=True, which='major', color='#666666', linestyle='-')

  plt.minorticks_on()
  plt.grid(b=True, which='minor', color='#999999', linestyle='-'
, alpha=0.2)
  plt.show()

# performing hold-out validation
# training the model and returning the
#  prediction accuracy with median_diff
```

Figure 60: Regression with hold-out validation (Part 1 of 2).

```python
def validate_model(model, features, targets):
  # splitting the data into
  #  training and testing features and predictions
  split = features.shape[0]//2
  train_features = features[:split]
  test_features = features[split:]

  train_targets = targets[:split]
  test_targets = targets[split:]

  # training the model
  model.fit(train_features, train_targets)

  # get the predicted_redshifts
  predictions = model.predict(test_features)

  # use median_diff function to calculate the accuracy
  return
 median_diff(test_targets, predictions), predictions, test_target
s



if __name__ == "__main__":
  data = np.load('sdss_galaxy_colors.npy')
  features, targets = get_features_targets(data)

  # initialize model
  dtr = DecisionTreeRegressor()

  # validate the model and print the med_diff
  diff, predictions, targets =
 validate_model(dtr, features, targets)
  print('Median difference: {:f}'.format(diff))

  print(predictions)
  print(targets)
  print(predictions.shape)
  print(targets.shape)
  plt.scatter(targets, predictions, s=0.4)
  plt.xlim((0, targets.max()))
  plt.ylim((0, predictions.max()))
  plt.xlabel('Measured Redshift')
  plt.ylabel('Predicted Recleardshift')
  plt.grid(b=True, which='major', color='#666666', linestyle='-')

  plt.minorticks_on()
  plt.grid(b=True, which='minor', color='#999999', linestyle='-'
, alpha=0.2)
  #plt.show()
```

Figure 61: Regression with hold-out validation (Part 2 of 2).

```python
import numpy as np
from matplotlib import pyplot as plt

# Complete the following to make the plot
if __name__ == "__main__":
    data = np.load('sdss_galaxy_colors.npy')
    # Get a colour map
    cmap1 = plt.get_cmap('YlOrRd')

    # Define our colour indexes u-g and r-i
    cidx_ug = data['u'] - data['g']
    cidx_ri = data['r'] - data['i']

    # Make a redshift array
    redshift = data['redshift']

    # Create the plot with plt.scatter and plt.colorbar
    plt.scatter(cidx_ug, cidx_ri, c=redshift, lw=0, s=1, cmap=
cmap1)
    colorbar = plt.colorbar().set_label("Redshift")

    # Define your axis labels and plot title
    plt.xlabel= "Colour index u-g"
    plt.ylabel= "Colour index r-i"
    plt.title = "Redshift (colour) u-g versus r-i"

    # Set any axis limits
    plt.axis([-0.5, 2.5, -0.5, 1])
    plt.grid(b=True, which='major', color='#666666', linestyle=
'-')
    plt.minorticks_on()
    plt.grid(b=True, which='minor', color='#999999', linestyle=
'-', alpha=0.2)
    #plt.legend()
    plt.show()
```

Figure 62: Colour vs Colour Redshift Plot.

```python
import numpy as np
from matplotlib import pyplot as plt,rc

from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeRegressor

rc('font',**{'family':'serif','serif':['Helvetica']})
rc('text', usetex=True)

def get_features_targets(data):
   #n lines, 4 columns
  features = np.zeros((data.shape[0], 4))
  features[:,0] = data['u'] - data['g']
  features[:,1] = data['g'] - data['r']
  features[:,2] = data['r'] - data['i']
  features[:,3] = data['i'] - data['z']
  targets = data['redshift']
  return (features, targets)

def median_diff(predicted, actual):
  diff = np.median(np.absolute(predicted - actual))
  return diff

def cross_validate_predictions(model, features, targets, k):
  kf = KFold(n_splits=k, shuffle=True)

  # declare an array for predicted redshifts from each iteration
  all_predictions = np.zeros_like(targets)

  for train_indices, test_indices in kf.split(features):
    print (train_indices, len(train_indices),test_indices, len(test_indices))
    # split the data into training and testing
    train_features, test_features = features[train_indices], features[test_indices]
    train_targets, test_targets = targets[train_indices], targets[test_indices]

    # fit the model for the current set
    model.fit(train_features, train_targets)

    # predict using the model
    predictions = model.predict(test_features)
```

Figure 63: K-Fold Cross-Validated Predictions (Part 1 of 2).

```python
 # put the predicted values in the
    # all_predictions array defined above
    all_predictions[test_indices] = predictions

    # note: the line above is filling
    # multiple disjoint elements of the
    #array at the same time. for example, if:
    #test_indexes is 8 11 23
    #with test_features z8, z11, z23
    #the code does: allpredictions[8,11,23] = z8, z11, z23
    #and hence, once per galaxy

  # return the predictions
  return all_predictions

if __name__ == "__main__":
  data = np.load('./sdss_galaxy_colors.npy')
  features, targets = get_features_targets(data)

  # initialize model
  dtr = DecisionTreeRegressor(max_depth=19)

  # call cross validation function
  predictions = cross_validate_predictions(dtr, features, targets, 10)

  # calculate and print the rmsd as a sanity check
  diffs = median_diff(predictions, targets)
  print('Median difference: {:.3f}'.format(diffs))

  # plot the results to see how well our model looks
  plt.scatter(targets, predictions, s=0.4)
  plt.xlim((0, targets.max()))
  plt.ylim((0, predictions.max()))
  plt.xlabel('Measured Redshift')
  plt.ylabel('Predicted Redshift')
  plt.grid(b=True, which='major', color='#666666', linestyle='-')

  plt.minorticks_on()
  plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
  plt.show()
```

Figure 64: K-Fold Cross-Validated Predictions (Part 2 of 2).

```python
import numpy as np
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeRegressor

import numpy as np
from matplotlib import pyplot as plt
from sklearn.tree import DecisionTreeRegressor

def get_features_targets(data):
  features = np.zeros((data.shape[0], 4)) #n lines, 4 columns
  features[:,0] = data['u'] - data['g']
  features[:,1] = data['g'] - data['r']
  features[:,2] = data['r'] - data['i']
  features[:,3] = data['i'] - data['z']
  targets = data['redshift']
  return (features, targets)

def median_diff(predicted, actual):
  diff = np.median(np.absolute(predicted - actual))
  return diff

# complete this function
def cross_validate_model(model, features, targets, k):
  kf = KFold(n_splits=k, shuffle=True)


# initialise a list to collect median_diffs for each iteration of the loop below

  mediandiffs = []

  for train_indices, test_indices in kf.split(features):
    train_features, test_features = features[train_indices], features[test_indices]
    train_targets, test_targets = targets[train_indices], targets[test_indices]

    # fit the model for the current set
    model.fit(train_features, train_targets)

    # predict using the model
    predictions = model.predict(test_features)


# calculate the median_diff from predicted values and append to results array

    mediandiffs.append(median_diff(test_targets, predictions))

  # return the list with your median difference values
  return mediandiffs
```

Figure 65: K-Fold Cross-Validation (Part 1 of 2).

```python


if __name__ == "__main__":
  data = np.load('./sdss_galaxy_colors.npy')
  features, targets = get_features_targets(data)

  # initialize model with a maximum depth of 19
  dtr = DecisionTreeRegressor(max_depth=19)

  # call your cross validation function
  diffs = cross_validate_model(dtr, features, targets, 10)

  # Print the values
  print('Differences: {}'.format(', '.join(['{:.3f}'.format(val) for val in diffs])))
  print('Mean difference: {:.3f}'.format(np.mean(diffs)))
```

Figure 66: K-Fold Cross-Validation(Part 2 of 2).

```
import numpy as np
from matplotlib import pyplot as plt, rc
from sklearn.tree import DecisionTreeRegressor

rc('font',**{'family':'serif','serif':['Helvetica']})
rc('text', usetex=True)


def get_features_targets(data):
  features = np.zeros((data.shape[0], 4)) #n lines, 4 columns
  features[:,0] = data['u'] - data['g']
  features[:,1] = data['g'] - data['r']
  features[:,2] = data['r'] - data['i']
  features[:,3] = data['i'] - data['z']
  targets = data['redshift']
  return (features, targets)

def median_diff(predicted, actual):
  diff = np.median(np.absolute(predicted - actual))
  return diff

def accuracy_by_treedepth(features, targets, depths):
  # split the data into testing and training sets
  split = features.shape[0]//2
  train_features = features[:split]
  test_features = features[split:]
  train_targets = targets[:split]
  test_targets = targets[split:]


# initialise arrays or lists to store the accuracies for the belo
w loop

  mediandiffs_training = []
  mediandiffs_test = []

  # loop through depths
  for depth in depths:
    # initialize model with the maximum depth.
    dtr = DecisionTreeRegressor(max_depth=depth)

    # train the model using the training set
    dtr.fit(train_features, train_targets)
```

Figure 67: Overfitting Trees with varying tree depth (Part 1 of 2).

```

# get the predictions for the training set and calculate their me
dian_diff

    predictions = dtr.predict(train_features)
    accuracy = median_diff(train_targets, predictions)
    mediandiffs_training.append(accuracy)


# get the predictions for the testing set and calculate their med
ian_diff

    predictions = dtr.predict(test_features)
    accuracy = median_diff(test_targets, predictions)
    mediandiffs_test.append(accuracy)

  # return the accuracies for the training and testing sets
  return (mediandiffs_training, mediandiffs_test)

if __name__ == "__main__":
  data = np.load('sdss_galaxy_colors.npy')
  features, targets = get_features_targets(data)

  # Generate several depths to test
  tree_depths = [i for i in range(1, 36, 2)]

  # Call the function
  train_med_diffs, test_med_diffs =
 accuracy_by_treedepth(features, targets, tree_depths)
  print("Depth with lowest median difference : {}"
.format(tree_depths[test_med_diffs.index(min(test_med_diffs))]))

  # Plot the results
  train_plot = plt.plot(tree_depths, train_med_diffs, label=
'Training set')
  test_plot = plt.plot(tree_depths, test_med_diffs, label=
'Validation set')
  hfont = {'fontname':'Helvetica'}
  plt.xlabel("Maximum Tree Depth", **hfont)
  plt.ylabel("Median of Differences", **hfont)
  plt.grid(b=True, which='major', color='#666666', linestyle='-')

  plt.minorticks_on()
  plt.grid(b=True, which='minor', color='#999999', linestyle='-'
, alpha=0.2)
  plt.legend()
  plt.show()
```

Figure 68: Overfitting Trees with varying tree depth (Part 2 of 2).

```
import numpy as np
import pydotplus as pydotplus
from sklearn.tree import DecisionTreeRegressor, export_graphviz
from IPython.display import Image

def get_features_targets(data):
    features = np.zeros(shape=(len(data), 4))
    features[:, 0] = data['u'] - data['g']
    features[:, 1] = data['g'] - data['r']
    features[:, 2] = data['r'] - data['i']
    features[:, 3] = data['i'] - data['z']
    targets = data['redshift']
    return features, targets


if __name__ == "__main__":
    data = np.load('sdss_galaxy_colors.npy')
    features, targets = get_features_targets(data)

    # Initialize model
    dtr = DecisionTreeRegressor(max_depth=3)
# We will come to this input in the next tutorial

    # Split the data into training and testing
    split_index = int(0.5 * len(features))
    train_features = features[:split_index]
    train_targets = targets[:split_index]

    dtr.fit(train_features, train_targets)

    dot_data = export_graphviz(dtr,
                               out_file=None,
                               feature_names=['u - g', 'g - r',
'r - i', 'i - z'],
                               rotate=False,
                               impurity=True,
                               label='all',
                               filled=True,
                               special_characters=True,
                               rounded=True
                               )

    graph = pydotplus.graph_from_dot_data(dot_data)
    Image(graph.write_png('d.png'))
```

Figure 69: Visualising decision tree regressor trained model with mean squared errors.

```
import numpy as np
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeRegressor
from matplotlib import pyplot as plt,rc
rc('font',**{'family':'serif','serif':['Helvetica']})
rc('text', usetex=True)

def get_features_targets(data):
  #n lines, 4 columns
  features = np.zeros((data.shape[0], 4))
  features[:,0] = data['u'] - data['g']
  features[:,1] = data['g'] - data['r']
  features[:,2] = data['r'] - data['i']
  features[:,3] = data['i'] - data['z']
  targets = data['redshift']
  return (features, targets)

def median_diff(predicted, actual):
  diff = np.median(np.absolute(predicted - actual))
  return diff

def cross_validate_model(model, features, targets, k):
  kf = KFold(n_splits=k, shuffle=True)

# initialise a list to collect median_diffs for each iteration of
the loop below

  mediandiffs = []

  for train_indices, test_indices in kf.split(features):
    train_features, test_features =
 features[train_indices], features[test_indices]
    train_targets, test_targets =
 targets[train_indices], targets[test_indices]

    # fit the model for the current set
    model.fit(train_features, train_targets)

    # predict using the model
    predictions = model.predict(test_features)
    p.append(predictions)


# calculate the median_diff from predicted values and append to r
esults array

    mediandiffs.append(median_diff(test_targets, predictions))

  # return the list with your median difference values
  return mediandiffs

```

Figure 70: Quasars vs Galaxy redshift predictions (Part 1 of 2).

```python
def split_galaxies_qsos(data):
  # split the data into galaxies and qsos arrays
  galaxies = data[data['spec_class'] == b'GALAXY']
  qsos = data[data['spec_class'] == b'QSO']

  # return the seperated galaxies and qsos arrays
  return (galaxies, qsos)

def cross_validate_median_diff(data):
  features, targets = get_features_targets(data)
  dtr = DecisionTreeRegressor(max_depth=19)
  return np.mean(cross_validate_model(dtr, features, targets, 10
))

if __name__ == "__main__":
    data = np.load('./sdss_galaxy_colors.npy')

    # Split the data set into galaxies and QSOs
    galaxies, qsos= split_galaxies_qsos(data)

    # Here we cross validate the model
    #  and get the cross-validated median difference

# The cross_validated_med_diff function is in "written_functions"

    galaxy_med_diff = cross_validate_median_diff(galaxies)
    qso_med_diff = cross_validate_median_diff(qsos)

    # Print the results
    print("Median difference for Galaxies: {:.3f}"
.format(galaxy_med_diff))
    print("Median difference for QSOs: {:.3f}"
.format(qso_med_diff))


# Median difference for Galaxies: 0.016
# Median difference for QSOs: 0.074
```

Figure 71: Quasars vs Galaxy redshift predictions (Part 2 of 2).

8.7 Morphology Classification

```python
import numpy as np
import math

def splitdata_train_test(data, fraction_training):

  # randomizing data set order
  np.random.seed(0)
  np.random.shuffle(data)

  # find split point
  training_rows = math.floor(len(data) * fraction_training)

  training_set = data[:training_rows]
  testing_set = data[training_rows:len(data)]
  return (training_set, testing_set)

if __name__ == "__main__":
  data = np.load('galaxy_catalogue.npy')


# setting the fraction of data which should be in the training se
t


  fraction_training = 0.7

  # spliting the data
  training, testing =
 splitdata_train_test(data, fraction_training)

  # printing the key values
  print('Number of galaxies in dataset:', len(data))
  print('Train fraction:', fraction_training)
  print('Number of galaxies in training set:', len(training))
  print('Number of galaxies in testing set:', len(testing))

  # print an example
  print('Field and value for 1 Galaxy:')
  for name, value in zip(data.dtype.names, data[0]):
    print('{:20} {:.6}'.format(name, value))
```

Figure 72: Splitting the galaxy catalogue.

```python
import numpy as np

def generate_features_targets(data):

  targets = data['class']
  #Create array where.
  #row = no. of trainng set record, column = 13
  features = np.empty(shape=(len(data), 13))

  features[:, 0] = data['u-g']
  features[:, 1] = data['g-r']
  features[:, 2] = data['r-i']
  features[:, 3] = data['i-z']
  features[:, 4] = data['ecc']
  features[:, 5] = data['m4_u']
  features[:, 6] = data['m4_g']
  features[:, 7] = data['m4_r']
  features[:, 8] = data['m4_i']
  features[:, 9] = data['m4_z']

  # filling the remaining 3 columns with
  # concentrations in the u, r and z filters
  features[:, 10] = data['petroR50_u']/data['petroR90_u']
  features[:, 11] = data['petroR50_r']/data['petroR90_r']
  features[:, 12] = data['petroR50_z']/data['petroR90_z']

  return features, targets


if __name__ == "__main__":
  data = np.load('galaxy_catalogue.npy')

  features, targets = generate_features_targets(data)

  # Print the shape of each array to check
  # the arrays are the correct dimensions.
  print("Features shape:", features.shape)
  print("Targets shape:", targets.shape)
```

Figure 73: Generating features and targets.

```python
import numpy as np
import math
from sklearn.tree import DecisionTreeClassifier


def splitdata_train_test(data, fraction_training):
  #randomizing data set order
  np.random.seed(0)
  np.random.shuffle(data)

  #finding split point
  training_rows = math.floor(len(data) * fraction_training)

  #splitting the data
  training_set = data[0:training_rows]
  testing_set = data[training_rows:len(data)]
  return (training_set, testing_set)

def generate_features_targets(data):
  targets = data['class']

  features = np.empty(shape=(len(data), 13))
  features[:, 0] = data['u-g']
  features[:, 1] = data['g-r']
  features[:, 2] = data['r-i']
  features[:, 3] = data['i-z']
  features[:, 4] = data['ecc']
  features[:, 5] = data['m4_u']
  features[:, 6] = data['m4_g']
  features[:, 7] = data['m4_r']
  features[:, 8] = data['m4_i']
  features[:, 9] = data['m4_z']

  # filling the remaining 3 columns with
  # concentrations in the u, r and z filters
  features[:, 10] = data['petroR50_u']/data['petroR90_u']
  features[:, 11] = data['petroR50_r']/data['petroR90_r']
  features[:, 12] = data['petroR50_z']/data['petroR90_z']

  return features, targets

```

Figure 74: Training the decision tree classifier (Part 1 of 2).

```python
def split_galaxies_qsos(data):
  # split the data into galaxies and qsos arrays
  galaxies = data[data['spec_class'] == b'GALAXY']
  qsos = data[data['spec_class'] == b'QSO']

  # return the seperated galaxies and qsos arrays
  return (galaxies, qsos)

def cross_validate_median_diff(data):
  features, targets = get_features_targets(data)
  dtr = DecisionTreeRegressor(max_depth=19)
  return np.mean(cross_validate_model(dtr, features, targets, 10
))

if __name__ == "__main__":
    data = np.load('./sdss_galaxy_colors.npy')

    # Split the data set into galaxies and QSOs
    galaxies, qsos= split_galaxies_qsos(data)

    # Here we cross validate the model
    #  and get the cross-validated median difference

# The cross_validated_med_diff function is in "written_functions"

    galaxy_med_diff = cross_validate_median_diff(galaxies)
    qso_med_diff = cross_validate_median_diff(qsos)

    # Print the results
    print("Median difference for Galaxies: {:.3f}"
.format(galaxy_med_diff))
    print("Median difference for QSOs: {:.3f}"
.format(qso_med_diff))


# Median difference for Galaxies: 0.016
# Median difference for QSOs: 0.074
```

Figure 75: Training the decision tree classifier (Part 2 of 2).

```python
import numpy as np
from matplotlib import pyplot as plt, rc
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict
from sklearn.tree import DecisionTreeClassifier
import itertools
from support_functions import plot_confusion_matrix,
generate_features_targets
rc('font', **{'family': 'serif', 'serif': ['Helvetica']})
rc('text', usetex=True)


def calculate_accuracy(predicted, actual):
    correct = 0
    # iterating over the two lists simultaneously
    for p, a in zip(predicted, actual):
        if p == a:
            correct += 1
    accuracy = correct / len(actual)
    return accuracy


if __name__ == "__main__":
    data = np.load('galaxy_catalogue.npy')

    # splitting the data
    features, targets = generate_features_targets(data)

    # training the model to get predicted and actual classes
    dtc = DecisionTreeClassifier()
    predicted = cross_val_predict(dtc, features, targets, cv=10)

    # calculating the model score using your function
    model_score = calculate_accuracy(predicted, targets)
    print("Our accuracy score:", model_score)

```

Figure 76: Accuracy and model's score with confusion matrix.

```python

 # calculating the models confusion matrix
 #  using sklearns confusion_matrix function
 class_labels = list(set(targets))
 model_cm = confusion_matrix(
     y_true=targets,
     y_pred=predicted,
     labels=class_labels)

 # Plot the confusion matrix
 #  using the provided functions.
 plt.figure()
 plot_confusion_matrix(
     model_cm,
     classes=class_labels,
     normalize=False)
 plt.savefig('confusion_matrix.png')
 plt.show()
```

Figure 77: Accuracy and model's score with confusion matrix.

```python
import numpy as np
from matplotlib import pyplot as plt,rc
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict
from sklearn.ensemble import RandomForestClassifier
from support_functions import
 generate_features_targets,plot_confusion_matrix, calculate_accur
acy

rc('font', **{'family': 'serif', 'serif': ['Helvetica']})
rc('text', usetex=True)

def rf_predict_actual(data, n_estimators):
    # generate the features and targets
    features, targets = generate_features_targets(data)

    # instantiate a random forest
    # classifier using n estimators
    rfc = RandomForestClassifier(n_estimators=n_estimators)

    # get predictions using 10-fold
    # cross validation with cross_val_predict
    predicted = cross_val_predict(rfc, features, targets, cv=10)

    # return the predictions and their actual classes
    return predicted, data['class']


if __name__ == "__main__":
    data = np.load('galaxy_catalogue.npy')

    # get the predicted and actual classes
    number_estimators = 90 # Number of trees
    predicted, actual =
 rf_predict_actual(data, number_estimators)

    # calculate the model score using your function
    accuracy = calculate_accuracy(predicted, actual)
    print("Accuracy score:", accuracy)

    # calculate the models confusion matrix
    #  using sklearns confusion_matrix function
    class_labels = list(set(actual))
    model_cm = confusion_matrix(
        y_true=actual, y_pred=predicted, labels=class_labels)

    # plot the confusion matrix using the
    #  provided functions.
    plt.figure()
    plot_confusion_matrix(model_cm, classes=class_labels,
normalize=False)
    plt.show()
```

Figure 78: Random Forest with confusion matrix.