Novel Applications of Machine Learning in Astronomy and Beyond

Ben Henghes

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London

Department of Physics and Astronomy

University College London

February 2, 2022

I, Ben Henghes, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

The field of astronomy is currently experiencing a period of unprecedented expansion, predominantly brought about by the vast amounts of data being produced by the latest telescopes and surveys. New methods will be required to have any hope of being able to analyse the data collected, the most widespread of which is machine learning. Machine learning has evolved rapidly over the past decade in an attempt to match the rate of increasing data, and aided by advancements in computer hardware, analyses that would have been impossible in the past are now common place on astronomers' laptops. However, despite machine learning becoming a favourite tool for many, there is often little consideration for which algorithms are best suited for the job.

In this thesis, machine learning is implemented in a variety of different problems ranging from Solar System science and searching for Trans-Neptunian Objects (TNOs), to the cosmological problem of obtaining accurate photometric redshift (photo-z) estimations for distant galaxies. In chapter 2 I implement many different machine learning classifiers to aid the Dark Energy Survey's search for TNOs, comparing the classifiers to find the most suitable, and demonstrating how machine learning can provide significant increases in efficiency. In chapter 3 I implement machine learning algorithms to provide photo-z estimations for a million galaxies, using the method as an example for how it is possible to benchmark machine learning algorithms to provide information about the scalibility of different methods. In chapter 4 I expand upon the benchmarking of methods developed for obtaining photo-z estimates, applying them instead to deep learning algorithms which directly use image data, before discussing future work and concluding in chapter 5.

Impact Statement

The work presented in this thesis describes applying machine learning methods in novel ways for different problems in astronomy. In the approaching era of enormous datasets produced by sky surveys such as the Vera C. Rubin Observatory's Legacy Survey of Space and Time (LSST), the Square Kilometre Array (SKA), and the Roman Space Telescope, machine learning will be vital in addressing many of the challenges which will come with the wealth of data being produced.

The work discussed in chapter 2 has immediate impact on the ability to detect Trans-Neptunian Objects (TNOs) using the Dark Energy Survey (DES), with the extra machine learning preprocessing stage allowing the detection pipeline to be run five times faster. Extensions of this work could allow for even greater performance boosts when applied to closer populations of TNOs or other even closer objects, such as asteroids. Furthermore, the work could be applied to surveys other than DES and the general problem of searching for rare events is widespread in many different disciplines with the machine learning process being easily transferable.

Benchmarking, such as is presented in chapter 3, is yet to be commonplace in research settings. While the benefits of having efficient and scalable models is obvious, they are often overlooked for small increases in accuracies, sometimes resulting in models which are hundreds of times slower and require far more computational resources for only the tiniest improvement in error. By encouraging more benchmarking to be carried out as part of the research, many of these computational resources could be freed-up, allowing for more research to be conducted and lowering the impact (both cost and environmental) of running computer clusters.

Finally, the deep learning models implemented in chapter 4 are state of the art convolutional neural networks (CNNs) and include never-before used mixed-input models which are applied to the problem of photometric redshift (photo-z) estimation. Photo-z estimates are a precursor of many cosmological experiments and providing a new, fast, and accurate model which could be used directly with images from surveys is invaluable. Indeed, from the comparisons we present, we found that the mixed-input inception CNN was able to perform between $30-50\%$ better than a traditional random forest, and was among the best performance found of any photo-z code in the literature.

Acknowledgements

To begin I must thank my supervisor, Ofer Lahav, not only for guiding my research from MSci to PhD and providing ideas and feedback, but also for being the perfect example of what a scientist and academic should be. Always asking questions, his staunch intellect would be intimidating but for his unmatched kindness in what can otherwise be a harshly competitive and lonely environment.

There are many others, both at UCL and further afield, who have helped steer my research in the right direction. In particular, I want to thank Ingo Waldmann who stepped in to become my second supervisor and for providing very useful comments and feedback for all my projects; and Jeyan Thiyagalingam who helped mentor me and was the best example of what collaborations across disciplines can accomplish.

I also want to thank Benjamin Joachimi and Suzanne Aigrain for agreeing to be my examiners, and providing the very detailed feedback which has helped improve this thesis.

Furthermore, I'm grateful to David Gerdes who initiated the project that led to my master's which ultimately resulted in me continuing to PhD; and Duncan Little, who was an excellent industry partner, greatly improving my coding ability and introducing me to the world of commercial research.

I would be remiss not to also mention the friends who made my time at UCL far more enjoyable. Indeed without their encouragement I would have been lost and hardly experienced the world I saw as being out of my reach. Thank you for taking me out of my comfort-zone, for helping me try new things and meet new people without whom I would have never made it this far.

My family have always been nothing but supportive, encouraging me to chase my dreams and allowing me the opportunities to do so. From a very young age I was shown the importance of education and my brother, Leo, invariably set the example for me to follow, imitate, and attempt to one-up. He’s been my role model since birth and through his continued fervour to do what he believes in, will forever be a role model I look-up to.

I can never thank my parents enough. They raised me, provided me with everything, and have taught me so much. Nothing I do would be possible without them and I hope they know how much I appreciate and love them.

Finally, Vanessa. Thank you for everything you’ve done these past four years. You’ve made my time and life immeasurably better. I love you.

Contents

List of Figures

List of Tables

"Many things that seem threatening in the dark become welcoming when we shine a light on them."

– Uncle Iroh

Chapter 1

Introduction

Throughout human history, as a species we have tried to better understand our place in the universe. From our earliest ancestors, who would have looked at the sky with wonder, the fascination with the heavens has been embedded in all of us. While there may have been those who began to formulate models of the solar system in ancient Greece, it was only in the 17^{th} century when we gathered more significant evidence that the Earth was not at the centre of the universe.

With the invention of the telescope and advent of observational astronomy, we were finally able to see further and gather more information to begin to understand the structure of the solar system. However, it took over a hundred years before William Herschel and other astronomers began to understand that the Sun also couldn't be at the centre of the universe, but rather was part of a larger structure, our Milky Way galaxy. It took hundreds more years before advances in optics allowed for Edwin Hubble to definitively classify Andromeda as a galaxy and prove that the universe was much larger than initially conceived (Hubble, 1929), with the Milky Way and Andromeda being but two of billions of galaxies in the universe.

Now, with the help of modern telescopes, we have finally started to gain a greater insight into the scale of the universe, and recent observations have suggested that there are likely around 200 billion galaxies in the observable universe (Lauer et al., 2021). Similar to how advances in technologies have allowed for larger, more advanced telescopes to be built (and launched into space) which have pushed forward our knowledge of the universe, modern astronomy has relied on parallel advancements in computing.

Analyses which would have been inconceivable just a few decades ago have been made possible thanks to the progress made in building faster processing units and larger memory stores. In particular, this allowed for the emergence of machine learning as one of the most powerful tools available in astronomical research. Machine learning methods are especially exciting in the current era of astronomy where the vast amount of data produced by large sky surveys and space telescopes would be impossible to analyse without specialist algorithms.

In this thesis I aim to contribute to the field of astronomy by making use of machine learning, testing and implementing different algorithms for a variety of problems to show their potential. The layout is as follows: In this chapter I introduce and provide background material to the various topics explored in the later chapters. Chapter 2 details the work completed in using machine learning to search for trans-Neptunian objects (TNOs) using the Dark Energy Survey (DES). In chapter 3 I focus on the benchmarking and scalability of machine learning algorithms applied to the problem of obtaining photometric redshift (photo-z) estimations of galaxies, and chapter 4 expands on this problem of obtaining photo-z estimates by using deep learning algorithms directly with image data.

Finally, I summarise and conclude the work in chapter 5, as well as providing appendices with additional details of the work completed as part of University College London's (UCL) centre for doctoral training in data intensive science (CDT-DIS) which did not directly relate to astronomy. This included a group project carried out over three months, and a placement completed in six months with the company ASOS, as well as a project related to the Covid-19 pandemic. In this final research project I investigated the correlations between natural UV radiation and Covid-19 cases in the UK.

1.1 Solar System Science

Despite the Solar System being our closest corner of the Universe, we still know very little about our 'cosmic backyard'. While much progress has been made since Galileo fathered observational astronomy at the beginning of the 17th century, it's only very recently that we have begun to build a more complete picture of the Solar System.

1.1.1 Formation and Evolution

One of the most fundamental questions asked about the Solar System is how it formed, and what dynamical processes have lead to the Solar System we observe today. The most widely accepted model of Solar System formation is the nebular hypothesis which was initially discussed in the 18th century by the philosophers Emanuel Swedenborg and Immanuel Kant (1755) before being refined by Laplace (1799). In the nebular hypothesis the Sun formed from a collapsing giant cloud of molecular gas. As the gas cloud begins with some angular momentum, the infall of gas results in the acceleration of the rotation of gas through the conservation of angular momentum. This forces the gas to spread out and form a protoplanetary disk around the core, with more gas then accreting onto the core from the disk.

The much more recent model for Solar System evolution is the Nice model (Tsiganis et al., 2005, Morbidelli et al., 2005, Gomes et al., 2005). The Nice model and its subsequent modifications describe how the the giant planets formed much nearer the Sun in the protoplanetary disk before migrating and settling into their current orbits. In this model the scattering of smaller planetesimals inwards (towards the Sun) by the giant planets resulted in angular momentum being transferred which caused the three outer planets, Saturn, Neptune and Uranus, to migrate outwards. In contrast, when planetesimals interacted with Jupiter they were sent into highly elliptical orbits or even ejected from the Solar System which then resulted in Jupiter migrating inwards.

A period of instability was caused when Jupiter and Saturn crossed their 2:1 mean motion resonance (MMR) - when Saturn would orbit the Sun twice for every one of Jupiter's orbits. As well as being a likely cause of the so called Late Heavy

Bombardment of the inner Solar System, this also resulted in Jupiter and Saturn shifting into their current orbits which in turn propelled Neptune and Uranus into more distant, eccentric orbits as well as swapping their order. Eventually the orbits of Uranus and Neptune stabilised through dynamical friction, where the remaining planetesimals in the disk were accelerated and as a result damped the eccentricity of the outer planets' orbits.

While the Nice model has done incredibly well to explain the evolution of the Solar System there is still much which has been left unanswered. The Nice model is especially prone to misrepresenting the outer Solar System, failing to predict the observed population of trans-Neptunian objects (TNOs). In particular there are now many of these observed objects orbiting further than Neptune with orbits too distant to have previously interacted with the outer planets, and a possible clustering of objects in their arguments of perihelia and longitude of ascending node cannot be explained.

Even with various modifications to the Nice model and experiments which have tried to more accurately replicate the population of the observed Solar System (Nesvorn̂y & Morbidelli, 2012), questions remain over the origins of the observed populations of the minor bodies of the Solar System. By observing more of these objects and learning more about the various populations we hope to discover more about their formation in the protoplanetary disk and what processes in their evolution have led to their current situations in the Solar System.

1.1.2 The Kuiper Belt

The Kuiper belt is the region of the Solar System past Neptune's orbit at around 30 AU where many minor bodies can be found. First theorised to exist following the discovery of Pluto, it was Edgeworth who began to try to quantify the number of planetesimals contained in the belt (Edgeworth, 1943) rather than Kuiper (who later suggested that a belt likely existed in the early Solar System but would have been cleared by Pluto which he believed to be far more massive) (Kuiper, 1951). It took almost half a century before telescopes became powerful enough to begin to discover more objects and confirm the existence of the belt.

Similar to the asteroid belt, the Kuiper belt (or Edgeworth-Kuiper belt) is mainly formed of small planetesimals which remained after the Solar System formed, however, it is far larger and about 100$\times$ more massive (Krasinsky et al., 2002, Pitjeva & Pitjev, 2018). The objects contained in the Kuiper belt also differ greatly in composition to those found in the asteroid belt, with most Kuiper belt objects (KBOs) being predominantly made of frozen volatiles (ice) rather than the rock and metal of asteroids.

As it is not possible to know the exact conditions during the formation of the Solar System, the KBOs act as fossils, preserving information and allowing us to probe earlier and learn more about the formation and evolution of the Solar System. Through studying the outer Solar System we can understand more about the dynamical processes that have led to what we observe now, and similarly learn how other extrasolar systems may have developed.

1.1.3 Trans-Neptunian Objects

A trans-Neptunian object (TNO) is defined as being any astronomical object in the Solar System that is orbiting the Sun at a distance greater than that of Neptune's average orbital distance of 30 AU. The first, and most famous, TNO discovered was the dwarf planet Pluto which was observed by Clyde William Tombaugh at the Lowell Observatory in 1930 (Tombaugh, 1946). For almost half a century Pluto remained the only known TNO, and it wasn't until 1978 that Charon (Pluto's largest satellite) was discovered (Christy & Harrington, 1978).

It took a further decade before additional TNOs began to be identified and it wasn't until the 21st century when the number of TNO discoveries reached the milestone of 100. Recently, helped by wide field telescopic surveys such as the Dark Energy Survey (DES) (DES Collaboration, 2016), the Panoramic Survey Telescope and Rapid Response System (Pan-STARRS) (Kaiser et al., 2002), and the Outer Solar System Origins Survey (OSSOS) (Bannister et al., 2018), the number of known TNOs has been rapidly increasing and as of 2021 there have been around 3500 TNOs discovered.

There is plenty of motivation for continuing to search for additional TNOs;

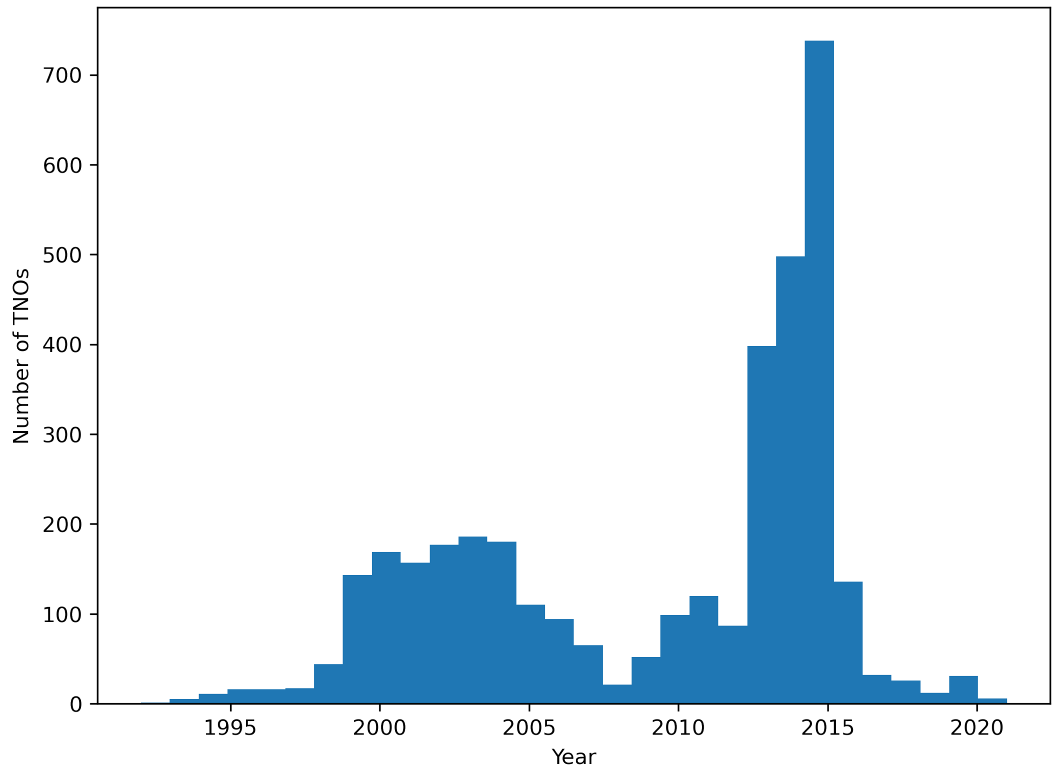

Figure 1.1: A histogram showing the distribution of TNOs discovered by year, with data provided by the International Astronomical Union's Minor Planet Center. The spike in detections from 2013-2015 was the result of observational runs of surveys such as DES, Pan-STARRS, and OSSOS.

in particular the orbital distribution of TNOs give a direct constraint on Neptune's evolution and migration. There have been many different models for how Neptune may have migrated (Ida et al., 2000, Tsiganis et al., 2005), however, it is still uncertain exactly what processes occurred. Objects in MMRs with Neptune provide the best evidence of whether Neptune's migration was 'jumpy' (Nesvornỳ & Vokrouhlickỳ, 2016), or 'smooth' (Nesvornỳ, 2015) as the number of TNOs found in each MMR depend on the model of migration. There have also been studies showing how some objects which were initially thought to have been scattered by Neptune could actually be in higher order MMRs (Hahn & Malhotra, 2005). This indicates that Neptune could have in fact migrated through a Kuiper belt which already contained objects with highly eccentric orbits.

With the discovery of more and more TNOs, various subclasses have emerged as a way of differentiating between the different populations based on their shared

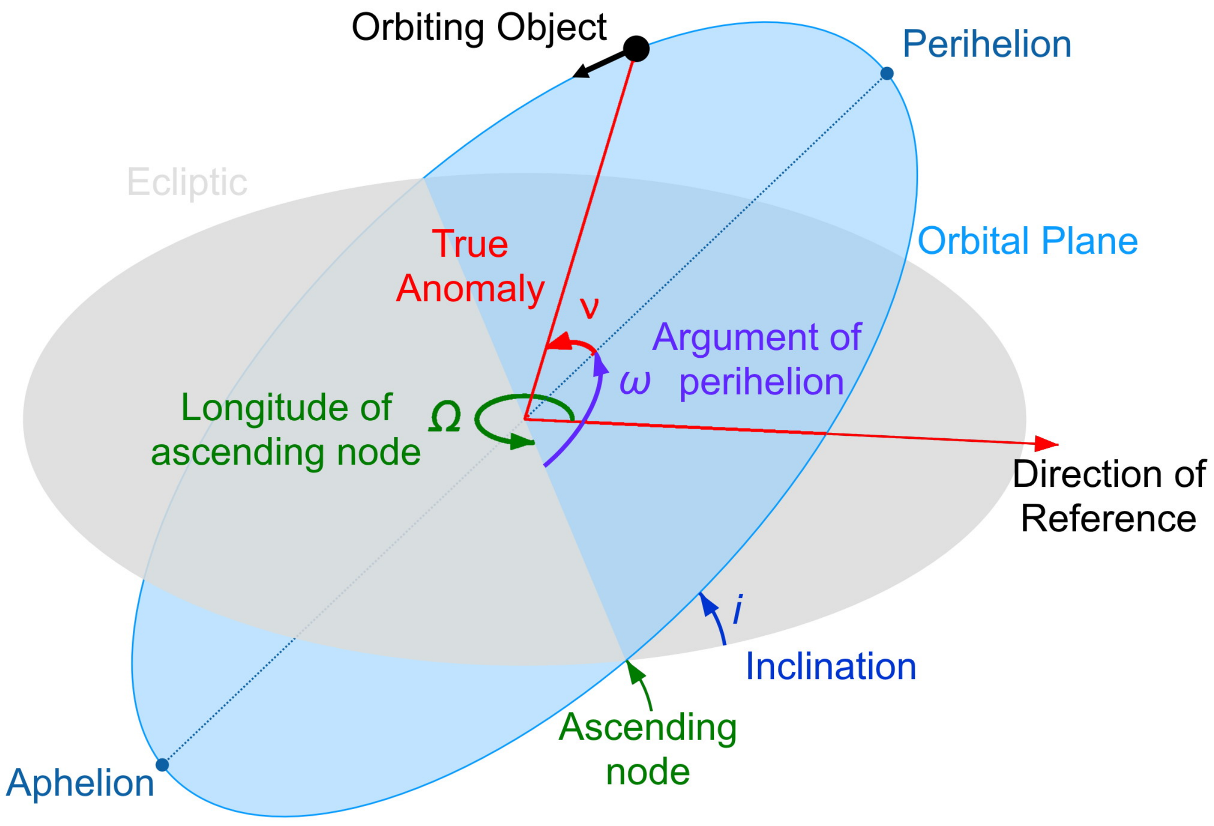

Figure 1.2: Figure showing the orbital elements, adapted from the image found at `https://commons.wikimedia.org/wiki/File:Orbit1.svg`

orbital characteristics. The orbits of Solar System bodies can be described by the following six parameters which are also shown in figure 1.2. The semi-major axis, a, and eccentricity, e, which describe the size and shape of the orbit. The inclination, i, gives the tilt of the orbit relative to the ecliptic plane (the average plane of the Solar System), and along with the longitude of ascending node, Ω, and argument of perihelion, ω, describes the full orientation of the system. Finally, the true anomaly, ν, is an angle that describes the position of the object along its orbit.

The true anomaly can also be related to two other angles: the mean anomaly, M, and the eccentric anomaly, E, which help to describe the orbit and are shown in figure 1.3. The mean anomaly represents the angle from perihelion of a fictitious body moving with a perfectly circular orbit, and hence constant speed, with the same period as the actual body in its real, elliptical orbit. This is particularly useful for describing orbits as if the mean anomaly is known at any given moment, it can then be calculated for any other epoch. The mean anomaly can be calculated from the eccentricity and eccentric anomaly using Kepler's equation:

$$M = E - e\sin(E), \tag{1.1}$$

which makes the eccentric anomaly a necessary prerequisite. The eccentric anomaly is defined as the angle between the perihelion, the centre of the elliptical orbit, and a fictitious point on the auxiliary circle encompassing the ellipse and perpendicular to the orbiting body. This allows it to be written in terms of the eccentricity and the true anomaly as

$$E = \cos^{-1}(\frac{e+\cos(\nu)}{1+e\cos(\nu)}). \tag{1.2}$$

There are also additional parameters which can be helpful when discussing TNOs such as the longitude of perihelion, $\bar{\omega}$, given by

$$\bar{\omega} = \Omega + \omega, \tag{1.3}$$

which is simply the sum of the longitude of ascending node and argument of perihelion and gives the orientation of the direction of perihelion. The perihelion distance, q, is another useful measure and can be easily obtained following Kepler's first law which allows q to be written in terms of the semi-major axis and eccentricity:

$$q = a(1-e). \tag{1.4}$$

This distance is regularly used to describe TNOs as while the semi-major axis can be very large, if the TNOs also have highly eccentric orbits, q can be very close to Neptune's orbital distance and so it can be used to help quickly identify how close their approach is.

TNOs are generally grouped into the following subcategories. Kuiper belt objects (KBOs), as previously mentioned, include all objects within the Kuiper belt.

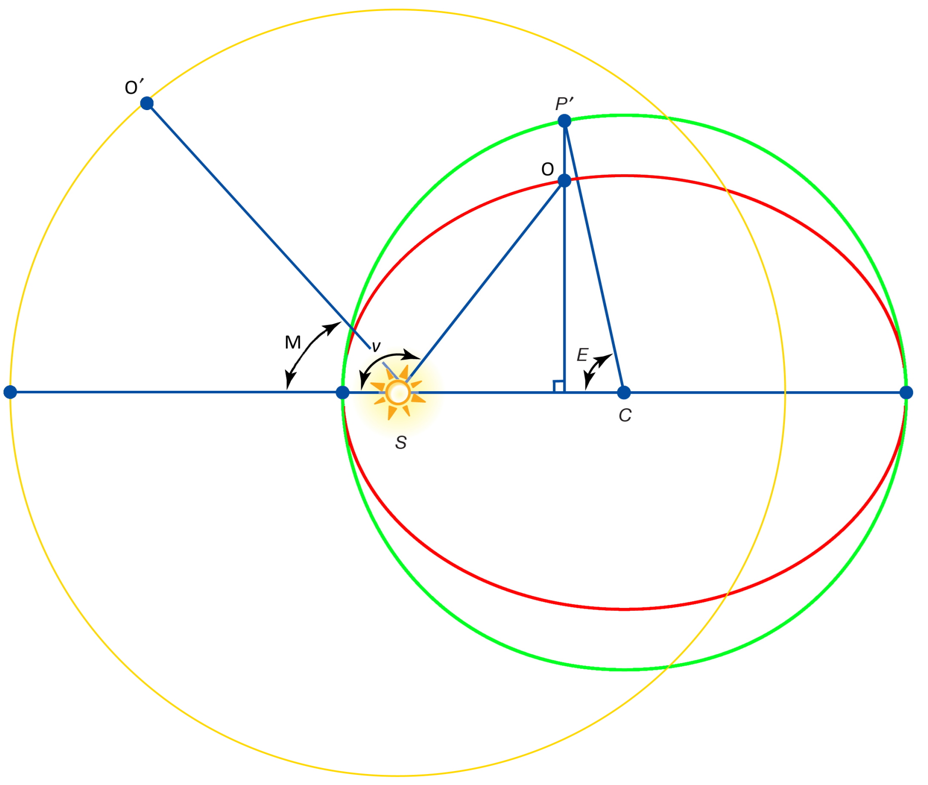

Figure 1.3: Figure showing the three anomalies used to describe orbits. The true anomaly, v, gives the angle which describes where an object, O, is in its orbit (red) about the Sun, S. The mean anomaly, M, gives the angle from perihelion of a fictitious object, O', moving in a circular orbit (yellow) with the same period as the original object (here the circular and elliptical orbits are not to scale). Finally, the eccentric anomaly, E, is shown to be defined as the angle between the Sun, the centre of the ellipse, C, and a fictitious point, P', on an auxiliary circle (green) perpendicular to the object.

These KBOs are often further classed as being either resonant TNOs, or classical KBOs. Resonant TNOs have orbital periods of an integer ratio to that of Neptune and include the group of 'plutinos' which share a 2:3 MMR with Neptune and are named after Pluto as the largest member. Classical KBOs instead have no resonance with Neptune and as such have less eccentric orbits and move on near circular orbits.

Scattered disk objects (SDOs) are another major group of TNOs which are classified by having highly eccentric, inclined orbits (Gomes et al., 2008). Unlike KBOs, SDOs are at risk of being disrupted by Neptune as their perihelia distances can be close to Neptune at around 30 AU. Indeed there have been investigations into whether the centaurs (minor bodies orbiting between the orbits of the giant planets) could in fact be SDOs which were forced further into the Solar System through interactions with Neptune (Horner et al., 2003).

'Extreme-TNOs' (ETNOs) are another way of separating the most distant TNOs, including any objects which have a semi-major axis $a > 150$ AU, and a perihelion distance $q > 30$ AU. As such they have a large overlap with SDOs, however, ETNOs also include detached objects which have larger perihelia distances and as such are unaffected by the gravitational forces from the giant planets. Finally, the most distant group of objects are the Sednoids, of which there are currently only three observed, with $q > 50$ AU keeping them completely outside of the Kuiper belt throughout their orbits (Sheppard et al., 2019).

1.1.4 Planet 9

Astronomers have been obsessed with searching for additional planets ever since the first planets were observed, and this zeal was only emphasised with the prediction and observation of Neptune due to perturbations in Uranus' orbit (Le Verrier, 1839, Adams, 1846, Galle, 1846). Many thought it would be possible to use the same technique of examining the perturbations seen in Neptune's orbit to discover additional planets (that would cause the perturbations through gravitational effects). Although Pluto was eventually discovered, it was only coincidental and not through any gravitational effects. Indeed, we now know that Pluto is so small that it can only be considered a minor planet, having not cleared its orbit and being one of many objects in the Kuiper belt.

While discovering any further large planets in the Solar System seemed unlikely, the recent detection of many TNOs has resulted in a renewed excitement in searching for a 'Planet 9'. This revived effort was caused by an observed grouping of orbital parameters in some ETNOs, shown in figure 1.4, which was first described by Trujillo & Sheppard (2014) with their detection of the TNO $2012VP_{113}$. It was noted that the ETNOs seemed to group in arguments of perihelia with, $\omega \approx 0^\circ$, and this could be caused by interactions with a large, distant planet. The similarity in orbital elements could be caused by the Kozai mechanism (or the Lidov-Kozai effect)

(Kozai, 1962, Lidov, 1961).

In the Kozai mechanism the TNOs act as a test particle, with negligible mass compared with the primary object - the Sun, and the more distant perturbing body - Planet 9. They would then have a conserved component of orbital angular momentum parallel to that of the Sun's and Planet 9's angular momentum. As the conserved angular momentum, L, can be expressed in terms of the eccentricity and inclination as

$$L = \sqrt{1-e^2}\cos(i) = constant, \tag{1.5}$$

an exchange between the eccentricity and inclination results in an oscillating argument of perihelia of the objects about a value of either $\omega = 0^\circ$ or $\omega = 180^\circ$.

However, this couldn't explain the observed orbits of all ETNOs, and a lack of objects with $\omega = 180^\circ$ resulted in alternative explanations being presented (de la Fuente Marcos & de la Fuente Marcos, 2014). It was Batygin & Brown (2016a) who first suggested that a Planet 9 could also cause similarities in the longitude of ascending node, Ω, through secular effects which could also then account for other highly inclined TNOs (Batygin & Morbidelli, 2017).

Despite the suggestions of how Planet 9 could help explain the observed TNO population as well as other properties such as the solar obliquity (Bailey et al., 2016) and the tilt of the invariable plane of the planets (Gomes et al., 2016), there is still no consensus on whether a planet is likely to exist at all. Instead, it has been suggested that the grouping of TNOs could be due to an observational bias (Bernardinelli et al., 2020a), and more observations are required to be able to sufficiently determine the significance of the observed grouping (Sheppard et al., 2019).

The Planet 9 hypothesis provided additional motivation for the work completed in chapter 2, where I implemented various machine learning algorithms to demonstrate how we could improve the efficiency of searching for TNOs and the hypothetical planet in DES (which, as described in section 1.3.2, is perfectly placed to detect these distant objects), with the possibility of discovering new objects to either strengthen the hypothesis or disprove it.

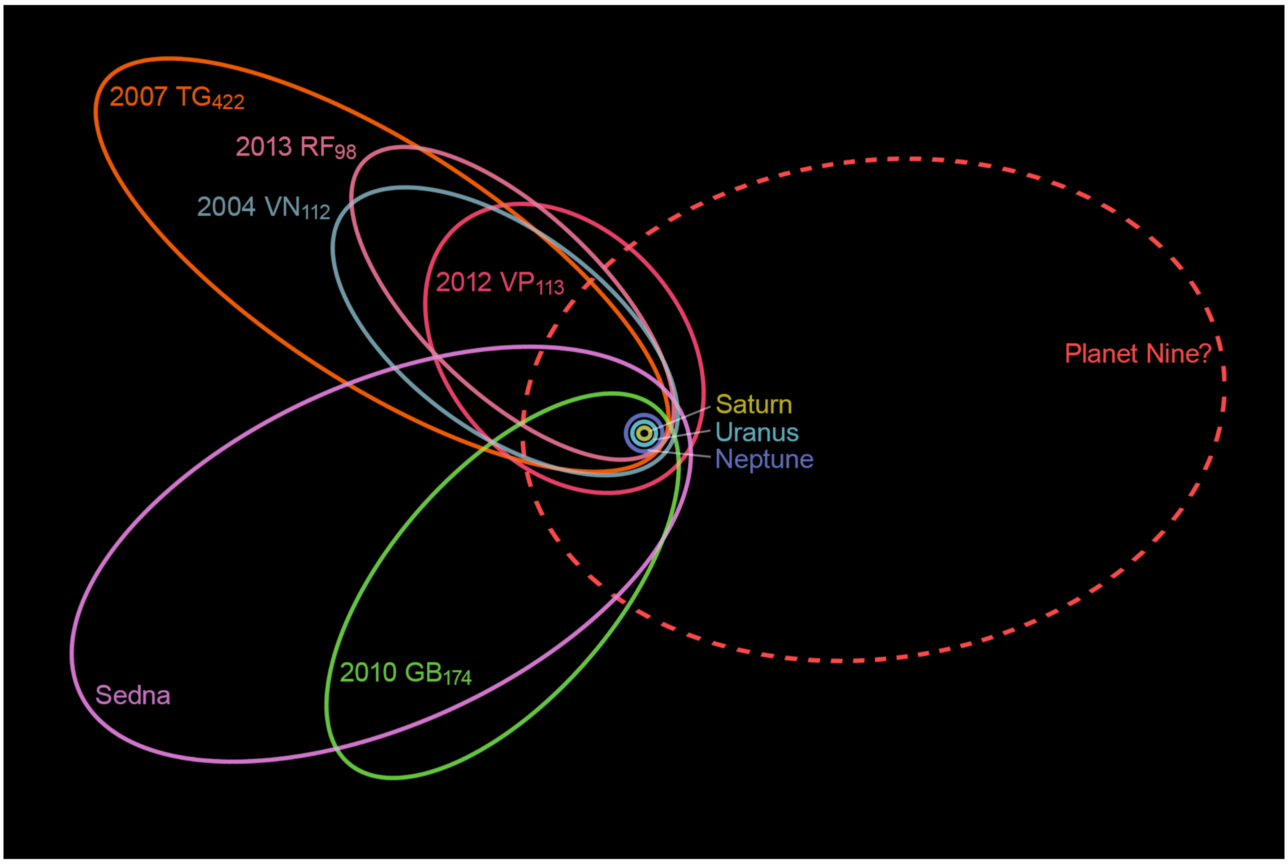

Figure 1.4: Figure showing the orbits of the initial ETNOs (including 2013RF$_{98}$, discovered by DES) which showed a clear grouping in their orbital parameters that was used to predict the existence of a 'Planet 9'. Image from `https://en.wikipedia.org/wiki/File:Planet_Nine_-_black_background.png`

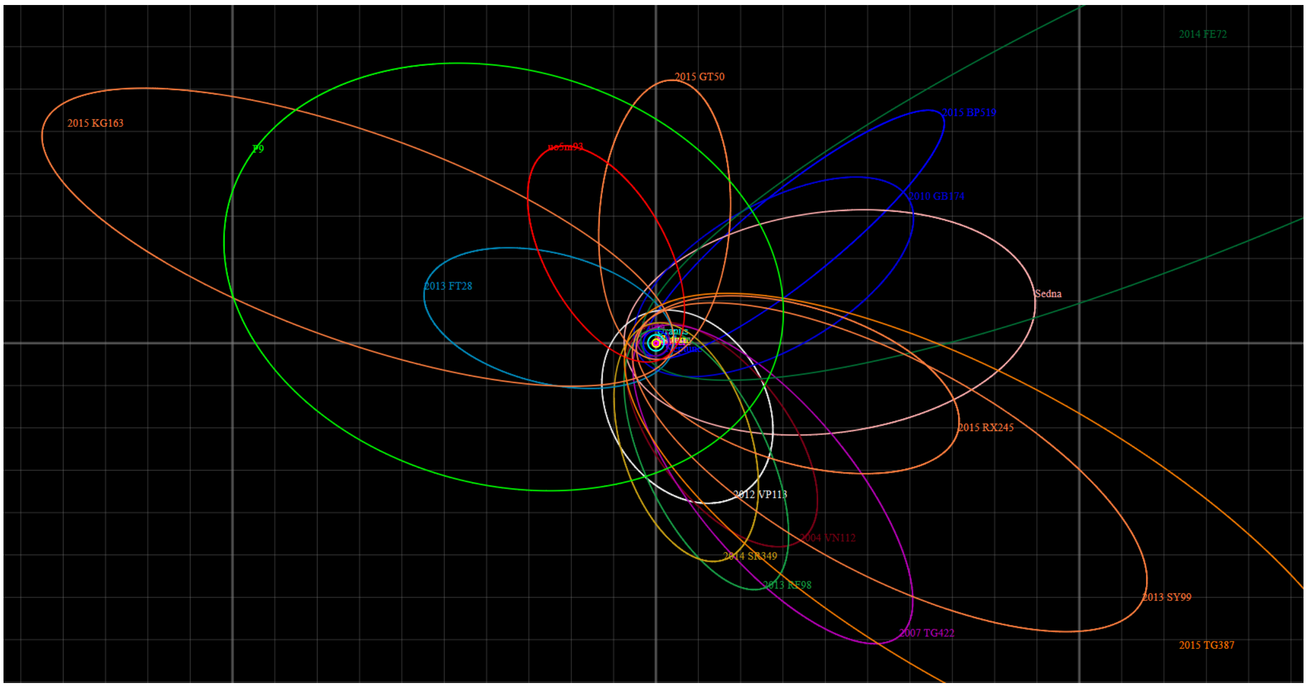

Figure 1.5: Figure showing the updated orbits of the additionally discovered ETNOs where a grouping in the longitude of ascending node is still visible, but less well defined. Image from `https://en.wikipedia.org/wiki/Planet_Nine#/media/File:Planet_nine-etnos_now-new3.png`

1.2 Cosmology

There have been many theories over the years trying to explain the universe we observe. This area of study, investigating the origin and evolution of the universe is the field of cosmology, and is one of the broadest areas of science. Here, I focus only on the area of physical cosmology relating to redshifts. The redshift is one of the most important properties in physical cosmology, describing the change in the wavelength of observed electromagnetic radiation, and is used in many different analyses. In this section I first give a very brief background about some of the main probes of cosmology: the cosmic microwave background, standard candles, large scale structure, and weak lensing, all of which provide motivation for finding redshifts, before I then describe the redshifts themselves and the methods for obtaining them.

1.2.1 Cosmological probes

1.2.1.1 Cosmic Microwave Background

The cosmic microwave background (CMB) discovered by Penzias & Wilson (1965) has been one of the most powerful probes for cosmology and gave direct evidence for the hot Big Bang model (Dicke et al., 1965). In the early universe, photons were tightly coupled to baryons; it was only as the universe expanded and cooled that the temperature dropped below a critical level and allowed neutral hydrogen to form during a period of recombination. This decoupled the photons which were no longer scattered by the free electrons, allowing them to travel freely through the universe and giving rise to the CMB observed today.

While the CMB is incredibly uniform, with a black body spectrum at temperature, $T \approx 2.7255$ K (Fixsen, 2009), the anisotropies provide key information about the conditions at recombination. By measuring these tiny anisotropies we can therefore learn a huge amount about various cosmological parameters. The latest data release from the Planck satellite (Planck Collaboration, 2020) provides the most detailed results to date about the CMB's angular power spectra and their measurements have shown consistency with the standard ΛCDM model of cosmology.

1.2.1.2 Standard Candles

Standard candles are astronomical objects for which the intrinsic luminosity is known. By measuring the observed luminosity one can then estimate the distance to the object. The first standard candles used were cepheid variables (Leavitt, 1908). These stars have oscillating luminosities which are directly correlated to the star's pulsation period. This allows astonomers to determine the intrinsic luminosity of the cepheid simply by measuring its pulsation period and hence find the distance to the cepheid. However, even with the latest telescopes, these objects can only be resolved for relatively close galaxies in the local universe, and for greater distances different candles are required.

Type Ia supernovae are supernovae thought to be caused by white dwarfs capturing material from a companion star. As the white dwarf accumulates enough matter for carbon fusion to occur, a runaway reaction results in the star going supernova. As the process is virtually the same in all type Ia supernovae, they have an observed peak luminosity that can then be related to the luminosity-distance (Betoule et al., 2014). Although this requires corrections in the shape of the light-curve and colour (lending them the name 'standardisable candles' rather than being a perfect standard candle), it still allows them to be used to infer distances to their host galaxies and makes them an invaluable rung on the cosmic distance ladder.

The distance ladder joins the different measurements of distances and allows for their calibration, connecting local geometric measurements (from parallax) to the more distant cepheid variables and type Ia suprnovae. The analysis of type Ia supernovae also famously led to the discovery of the acceleration of the expansion of the universe by Riess et al. (1998) and Perlmutter et al. (1999) which was awarded the Nobel prize in physics in 2011.

Finally, the detection of gravitational waves at the Laser Interferometer Gravitational-Wave Observatory (LIGO) (Abbott et al., 2016) has led to an extension of the distance ladder. The gravitational waves provide a new method to directly measure the luminosity distance from 'standard sirens', binary mergers of neutron stars or black holes (Schutz, 1986, Holz & Hughes, 2005). Binary neutron star mergers, such as the event GW170817, have optical counterparts lending them the name ‘bright sirens’ and can be used to determine the exact host galaxy (Abbott et al., 2017). Black hole mergers with no optical counterpart are instead labelled as ‘dark sirens’.

With a distance measured, the distance-redshift relation can then be used to infer cosmological parameters. This provides a great deal of motivation to measure accurate redshifts. For supernovae and bright sirens the host galaxies can be identified and the redshifts can be more easily obtained, however, it is also possible to use dark sirens as with a sufficient catalog of potential host galaxies the redshift distribution can be used instead (Soares-Santos et al., 2019).

1.2.1.3 Large Scale Structure

Large scale structure (LSS) refers to the structure of matter and galaxies in the universe on very large scales (from Mpc to Gpc). The observed distribution of galaxies in the universe is a natural probe of the universe’s matter distribution, and the redshifts of galaxies are required to map this distribution in three-dimensions. However, while the distribution of galaxies does trace the underlying matter distribution, it acts as a biased tracer of dark matter, not a direct measurement. Despite this, the observations of LSS allow for many cosmological measurements including redshift space distortions (RSD) and baryon acoustic oscillations (BAO).

Redshift space distortions arise due to a Doppler shift caused by the peculiar velocities of galaxies. Two distinct effects have been observed. The first, known as the ’Fingers of God’ effect, is the result of the random motions of galaxies in clusters and acts to elongate the redshift distribution (Jackson, 1972). The second, called the Kaiser effect, is caused by the infall of galaxies in an assembling cluster which instead acts to flatten the structure (Kaiser, 1987). As well as probing the structure and underlying matter distribution, measuring these effects in redshift surveys is also vital as the RSD distort the measured redshifts of galaxies and therefore change the inferred distances.

Baryon acoustic oscillations are periodic fluctuations of the baryonic matter density resulting from the conditions in the early universe. Prior to the period of recombination, when photons were still coupled to the baryonic matter, overdensities attracted matter gravitationally which resulted in an outward pressure due to Thomson scattering. These opposite forces created oscillating waves in the primordial plasma, until the universe cooled sufficiently for the decoupling of photons from the baryonic matter. This stopped the waves from propagating and caused the shells of baryons to be 'frozen' in place, with the maximum distance travelled of around 150 Mpc called the sound horizon.

As the matter in shells goes on to form galaxies, one therefore expects more galaxies to form with a separation distance equal to that of the sound horizon. This signal was detected by both the Sloan Digital Sky Survey (SDSS) (Eisenstein et al., 2005) and the Two-degree-Field Galaxy Redshift Survey (2dF) (Cole et al., 2005). Furthermore, the BAO can act as a 'standard ruler' allowing for a comparison of the sound horizon today, to that at the period of recombination (by observing the CMB). This allows it to provide a method independent to that of supernovae to measure the acceleration of the expansion of the universe.

1.2.1.4 Weak Lensing

Gravitational lensing has been a known effect since introduced by Einstein (1936). The effect of gravity of a mass in the line of sight of observation can distort images of background objects, with deep gravitational potential wells caused by objects such as galaxy clusters resulting in 'strong lensing', and more subtle effects caused by moderate perturbations resulting in 'weak lensing' (Bartelmann & Schneider, 2001). Weak lensing can then be used to measure the intervening matter, and unlike the distribution of galaxies in the universe which act as a trace of the matter distribution, weak lensing gives a direct measurement. However, one of the main prerequisites for weak lensing analyses is the redshift distribution of the galaxies whose light is being distorted, and hence one requires very accurate redshifts.

Uncertainties in the photometric redshifts of galaxies used in weak lensing analyses therefore have a great impact on the accuracies of the inferred cosmological parameters (Ma et al., 2006). More specifically, a positive photo-z bias results in a lower weak lensing signal due to underestimating the surface mass density;

however, the effect of the bias is more complex, not only being in its mean, but also being a function of the redshift. Typically the photo-z bias will be opposite for low and high redshift galaxies which can result in a smaller (or zero) mean bias, and while the biases may cancel out, their effects on the lensing signal does not. Similarly, the other major metrics of photo-z methods (the scatter and outlier fraction) may indicate a calibration bias, however, they are not sufficient to determine the efficacy of the photo-z measurements for lensing and custom metrics may be more optimised (Mandelbaum et al., 2008).

1.2.2 Redshifts

A redshift is defined as an increase in wavelength, or decrease in frequency and energy, of electromagnetic radiation. The redshift, z, can be written as the ratio between the relative change in wavelengths:

$$z = \frac{\lambda_o - \lambda_e}{\lambda_e} = \frac{\Delta\lambda}{\lambda_e}, \tag{1.6}$$

where λ_o is the observed, redshifted wavelength and λ_e is the emitted wavelength.

The scale factor, a, is another commonly used term in cosmology which describes how distances between points in the universe change as the universe expands and by definition has a value equal to one today. It can then be used to relate the emitted wavelength to the observed wavelength as

$$\lambda_e = a\lambda_o, \tag{1.7}$$

which allows us to write the relation between the redshift and the scale factor,

$$a = \frac{1}{1+z}. \tag{1.8}$$

This suggests that redshift can be used as an alternative way of quantifying the relative size of the universe. As an example, at $z = 1$, equation 1.8 shows that $a = \frac{1}{2}$, so the universe would have been half its current size. As the scale factor by definition is also linked to time, the redshift can then also be used to describe when events happened.

Locally, for $z << 1$, the redshift of galaxies can be equated to the Doppler shift caused by the galaxies' recession velocities. The relativistic Doppler effect can also be written as

$$\frac{\lambda_o}{\lambda_e} = \frac{1+\frac{v}{c}}{\sqrt{1-\frac{v^2}{c^2}}}, \tag{1.9}$$

where v is the velocity of the emitting source which is moving away from the observer, and c is the speed of light. As the motion of the source is generally much less than the speed of light, with $v << c$, equation 1.9 can be approximated to

$$\frac{\lambda_o}{\lambda_e} \approx 1+\frac{v}{c}, \tag{1.10}$$

which allows the redshift to be written as

$$z \approx \frac{v}{c}. \tag{1.11}$$

1.2.3 Spectroscopic Redshifts

The redshift is a directly observable quantity as it is possible to measure the observed wavelength of light through spectroscopy, and by comparing this to the wavelength of light at which it was known to be emitted, the redshift can be obtained using equation 1.6. The process of spectroscopy is much the same as when Newton first used a prism to demonstrate that sunlight was in fact a 'spectrum' of light which when passed through the prism produced a rainbow (Newton, 1672).

Modern spectrographs use fiber-optic cables with many fibres to pass light through a slit. The light is split using dichroics before being spread with a volume phase holographic (VPH) grating and imaged using high resolution cameras. This allows for the spectra to be measured with very high precision, and by comparing features of the redshifted spectra with known rest-frame spectra taken of various atoms and molecules on Earth, the redshift can be calculated.

Many surveys were designed to perform spectroscopy with the aim of obtaining spectroscopic redshifts of galaxies including the Baryon Oscillation Spectroscopic Survey (BOSS) (Dawson et al., 2012) and extended-BOSS (eBOSS) (Dawson et al., 2016). New surveys such as the Dark Energy Spectroscopic Instrument (DESI) (Martini et al., 2018) and the Roman space telescope (Spergel et al., 2015) aim to obtain millions more spectroscopic redshifts. However, wide field photometric surveys such as DES have observed hundreds of millions of galaxies (DES Collaboration, 2005), and with even larger suveys such as the Vera C Rubin Observatory's Legacy Survey of Space and Time (LSST) aiming to observe billions of galaxies (Tyson et al., 2003), it is simply not feasible to perform spectroscopy for every galaxy.

1.2.4 Photometric Redshift Estimation

Instead of performing spectroscopy, it is possible to estimate the redshift of galaxies by using photometry. The different wavelength bands of the filters used to image the galaxies can be thought of as a very sparse spectra which can then be used to obtain photometric redshift (photo-z) estimates. There are two methods widely used in photo-z estimation: template fitting (Benitez, 2000, Bolzonella et al., 2000), and machine learning (Collister & Lahav, 2004, Abdalla et al., 2011).

1.2.4.1 Template Fitting

Template fitting is the process of using a set of predefined galaxy spectra (the templates) and fitting the photometry of the galaxy to the templates, where the photo-z is then inferred from finding the best fitting template. It was first implemented by Puschell et al. (1982) who obtained photo-z estimates of faint radio galaxies using spectral energy distribution (SED) templates. The method was quickly improved by Loh & Spillar (1986) who were able to obtain photo-z estimates for thousands of galaxies and adapted the method into the template fitting used to this day.

The method can be visualised using figure 1.6, taken from the original Loh & Spillar (1986) paper. It shows the SED template, in this case an elliptical galaxy template from Bruzual et al. (1983), and how the flux of the galaxy is expected to change at varying redshifts. The measured flux of a different galaxy is shown as

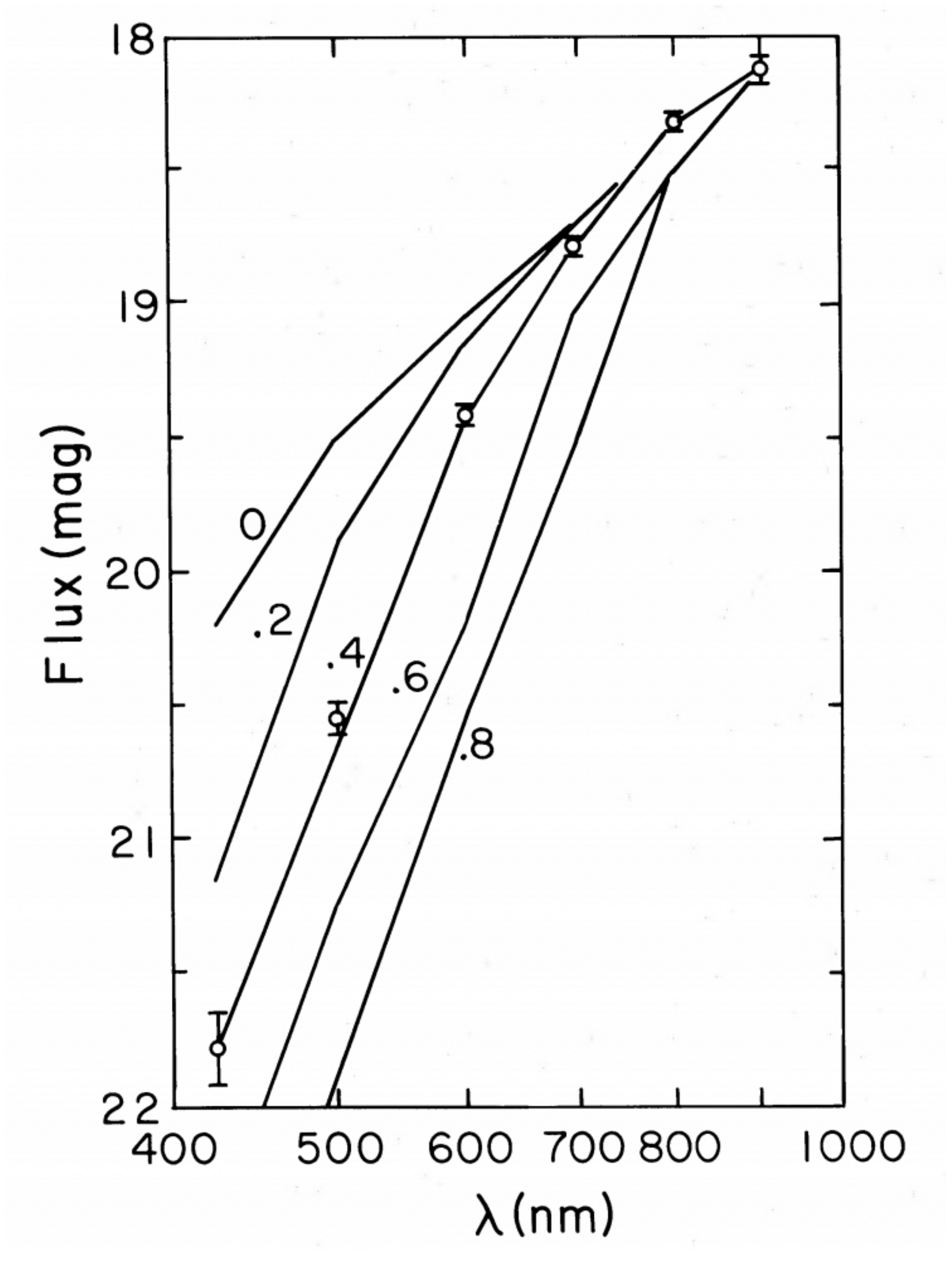

Figure 1.6: Figure showing the process of template fitting taken from Loh & Spillar (1986). The SED template of an elliptical galaxy shows how the flux is expected to change at varying labelled redshifts. The measured flux of the observed galaxy is shown as open circles and by matching it to the template, in this case close to the $z = 0.4$ line, one can obtain the photo-z estimate.

open circles and acts as an approximate measure of the underlying SED. As they are seen to lie close to the line at $z = 0.4$, it resulted in a photo-z estimate of $z = 0.398 \pm 0.018$ for this galaxy (which had a spectroscopic redshift of $z_{spec} = 0.390$).

Template fitting is a very intuitive model and more recent implementations have been able to incorporate physical information such as dust extinction, however, including these physical constraints also requires very careful calibration as well as an accurate model (Benitez, 2000).

1.2.4.2 Machine Learning

Machine learning has become the leading empirical method of photo-z estimation since Firth et al. (2003) first used neural networks to obtain photo-z estimates. There are many different supervised machine learning algorithms which can be applied to the problem of obtaining photo-z estimates (and I describe the most commonly used algorithms in section 1.4), but all of them follow the same principle. The algorithms require a large, representative training set, with the 'true' values of the redshift (in this case the spectroscopic redshift) already labelled. They then learn to create a mapping from selected input features of the data to the output redshift.

For traditional machine learning algorithms, such as Decision Trees, Random Forests, or k-Nearest Neighbours, the features are obtained from photometry, where the predominant features are the magnitudes measured using different filters which can also be combined into colours. However, deep learning methods which are increasingly being used take the image itself as the input, where the pixel values are akin to features (Hoyle, 2016). Exploring these methods forms the basis of chapter 3 where I use different traditional machine learning methods to demonstrate the merits of benchmarking applied to photo-z estimation, as well as chapter 4 where I use deep learning methods to obtain photo-z estimates directly from image data.

There are many benefits to using machine learning, but in general they are able to produce better overall photo-z estimates when compared with template fitting methods (Hildebrandt et al., 2010). When plotting the photo-z against spectroscopic redshift, machine learning methods usually produce plots with less scatter and a smaller outlier fraction, however, template fitting methods do perform better at the extremes, with better estimates near the upper and lower redshift limits (Abdalla et al., 2011).

Although machine learning methods can outperform template fitting methods,

Table 1.1: List of popular template fitting and machine learning photo-z codes.

Code	Authors	Type
BPZ	Benitez (2000)	Template Fitting
hyperz	Bolzonella et al. (2000)	Template Fitting
ANNz	Collister & Lahav (2004)	Machine Learning
Le Phare	Arnouts et al. (1999), Ilbert et al. (2006)	Template Fitting
ZEBRA	Feldmann et al. (2006)	Template Fitting
EAZY	Brammer et al. (2008)	Template Fitting
ArborZ	Gerdes et al. (2010)	Machine Learning
TPZ	Carrasco Kind & Brunner (2013)	Machine Learning
SkyNet	Graff et al. (2014)	Machine Learning
ANNz2	Sadeh et al. (2016)	Machine Learning
GPz	Almosallam et al. (2016)	Machine Learning
Delight	Leistedt & Hogg (2017)	Hybrid
Metaphor	Cavuoti et al. (2016)	Machine Learning
CMNN	Graham et al. (2018)	Machine Learning

they are limited by the training set. As the training set must be representative, machine learning models cannot be easily extrapolated, and as most galaxies with spectroscopic redshifts measured are brighter and at lower redshifts, the machine learning methods often struggle at higher redshift. Template fitting photo-z estimation is also limited by the templates available and it is assumed that the templates are well calibrated and representative. They are also less flexible when it comes to adding additional information, whereas machine learning models can easily accept additional features. However, when the number of features is significantly increased, the algorithms become more prone to overfitting. As such it could be beneficial to use template fitting and machine learning methods together (Leistedt & Hogg, 2017).

As machine learning has become more widely used for the problem of photometric redshift estimation, there have been many different codes created for the purpose of producing accurate photo-z estimates. Table 1.1 lists some of the most widely used codes including both machine learning and template fitting methods, and the comparisons conducted by Abdalla et al. (2011), Sánchez et al. (2014), and Schmidt et al. (2020) provide an overview of the current photo-z landscape.

1.3 Sky Surveys

Sky surveys have the general purpose of building a catalog of astronomical objects which can then be used for many different analyses. The first attempt of creating a survey was the Astrographic Chart (Turner, 1912) which used observatories around the world to map the stars. Beginning towards the end of the 19^{th} century, the positions of around 5 million stars were measured over several decades to create the largest catalog of the time. However, it wasn't until the first space mission designed specifically for astrometry, Hipparcos, when the measurements of the Astrographic Chart were most useful to derive the proper motions for all 2.5 million stars in the Tycho-2 catalog (Urban & Corbin, 1998).

In this thesis I worked with data from two different surveys. In chapter 2 I used Dark Energy Survey (DES) data to help aid the search for TNOs, and in chapters 3 and 4 I used Sloan Digital Sky Survey (SDSS) data to obtain photo-z estimates for over a million galaxies; where chapter 3 used magnitude features to benchmark different classical machine learning algorithms and chapter 4 used deep learning methods directly with image data.

Galaxy surveys such as SDSS and DES have the aim to measure as many galaxies as possible to calculate the cosmological parameters and allow for the large scale structure of the universe to be observed. Table 1.2 compares different galaxy surveys as well as surveys which are planned for the future. As can be seen from this table, the number of objects has rapidly increased from the first galaxy survey, the CfA Redshift Survey, which measured the spectra of galaxies one-by-one to obtain a catalog of around 18000 objects (Huchra et al., 1983). The 2dF Galaxy Redshfit Survey was able to use 400 optical fibres to measure 400 spectra simultaneously and obtained around a quarter of a million spectra (Colless et al., 2001), and SDSS similarly used 640 fibres originally (and has since upgraded to 1000 fibres) to obtain spectra for over 2 million galaxies (Alam et al., 2015). Furthermore, the Dark Energy Spectroscopic Instrument (DESI) has 5000 fibres to allow for millions more galaxy spectra to be collected (Martini et al., 2018).

However, despite this increase in the capability to measure galaxy spectra, the

Table 1.2: A table comparing different galaxy surveys which have been completed or planned, with notes on the number of total objects (including stars) observed and whether they are spectroscopic surveys, photometric surveys, or both.

Survey	**Dates of Observations**	**Number of Objects Observed (or Planned)**	**Notes**
CfA Redshift Survey	1977 - 1982 (CfA1), 1985 - 1995 (CfA2)	0.018 M	All with spectra
2dF Galaxy Redshift Survey	1997 - 2002	0.4 M	Including around 0.25 M galaxy spectra
Sloan Digital Sky Survey (SDSS)	2000 -	500 M	Inluding over 2 M galaxy spectra.
Dark Energy Survey (DES)	2013 - 2019	700 M	Only photometry
Dark Energy Spectroscopic Intrument (DESI)	2019 -	40 M	All with spectra
Vera C Rubin Observatory's LSST	2022 -	2000 M	Only photometry
Euclid	2022 -	1000 M	Will include 50 M galaxy spectra
Roman (previously WFIRST)	2025 -	1000 M	Will include photometry and spectra

next generation of surveys such as the Vera C Rubin Observatory's Legacy Survey of Space and Time (LSST) (Tyson et al., 2003), and the Euclid (Laureijs et al., 2011) and Roman (Spergel et al., 2015) space telescopes will observe billions of galaxies. As such spectroscopy cannot be relied on as the sole source of redshifts, and instead photometric redshifts will continue to be an area of great importance.

1.3.1 The Sloan Digital Sky Survey

SDSS (York et al., 2000) is one of the largest sky surveys with publicly available data. With a dedicated 2.5 m telescope located in the Sacramento mountains, New Mexico, at the Apache Point Observatory (Gunn et al., 2006) SDSS has observed over 200 million galaxies to date (Ahumada et al., 2020). As well as performing photometry for 200 million galaxies, the spectroscopic surveys completed as part of SDSS, the Baryon Oscillation Spectroscopic Survey (BOSS) (Dawson et al., 2012) and extended-BOSS (eBOSS) (Dawson et al., 2016) have measured around 2 million galaxy spectra in the area of the sky shown in figure 1.7.

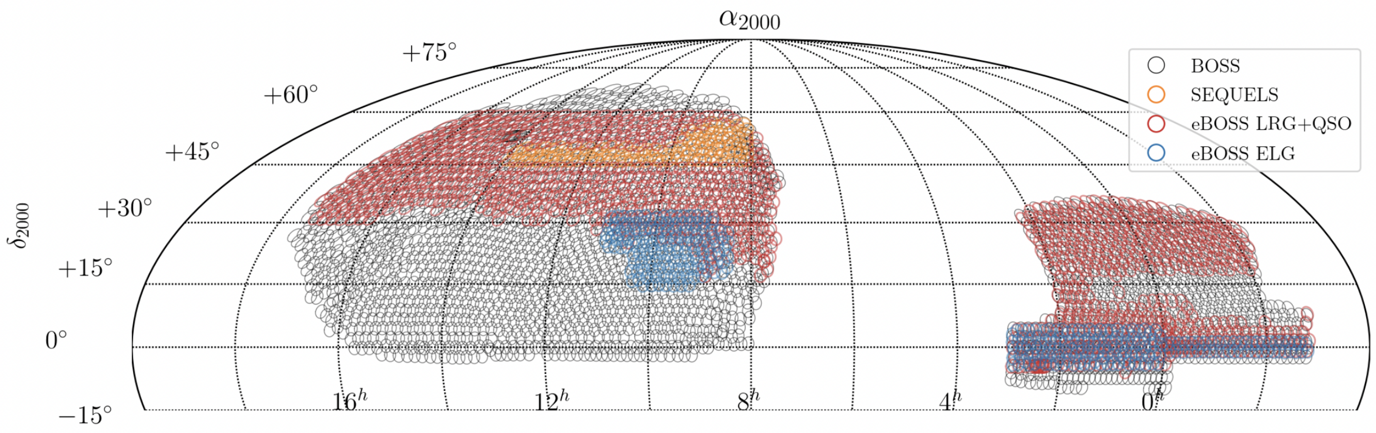

Figure 1.7: Figure showing the SDSS-DR16 eBOSS coverage in equatorial coordinates which was originally shown in the DR16 paper by Ahumada et al. (2020).

For the work completed in chapter 3 I downloaded the photometric magnitudes and spectroscopic redshifts of 1639348 galaxies. Using the five magnitudes from the five different optical filters (u, g, r, i, and z) as features, it was possible to obtain photo-z estimates which were then compared to the spectroscopic redshift to determine the performance of different machine learning algorithms. In chapter 4 I downloaded images for 1059678 galaxies (also in all five filters), as well as their corresponding magnitudes and spectroscopic redshifts. These images were then used to create a novel deep learning method for photo-z estimation.

1.3.2 The Dark Energy Survey

DES (DES Collaboration, 2005) is a wide field optical to near infrared survey which over the course of six years imaged 5000 square degrees of the southern sky, as shown in figure 1.8. Using the Dark Energy Camera (DECam) (Flaugher et al., 2015) on the 4 m Blanco telescope at the Cerro Tololo Inter-American Observatory (CTIO), Chile, DES observed hundreds of millions of galaxies with the primary goal of better constraining the dark energy equation of state (DES Collaboration, 2016).

While DES was designed as a cosmological survey, its repeat observations over a wide area of the sky made it an excellent source of data for studying Solar System objects as well (DES Collaboration, 2016). Indeed, DES and DECam have been used to discover many minor planets (Trujillo & Sheppard, 2014, Gerdes

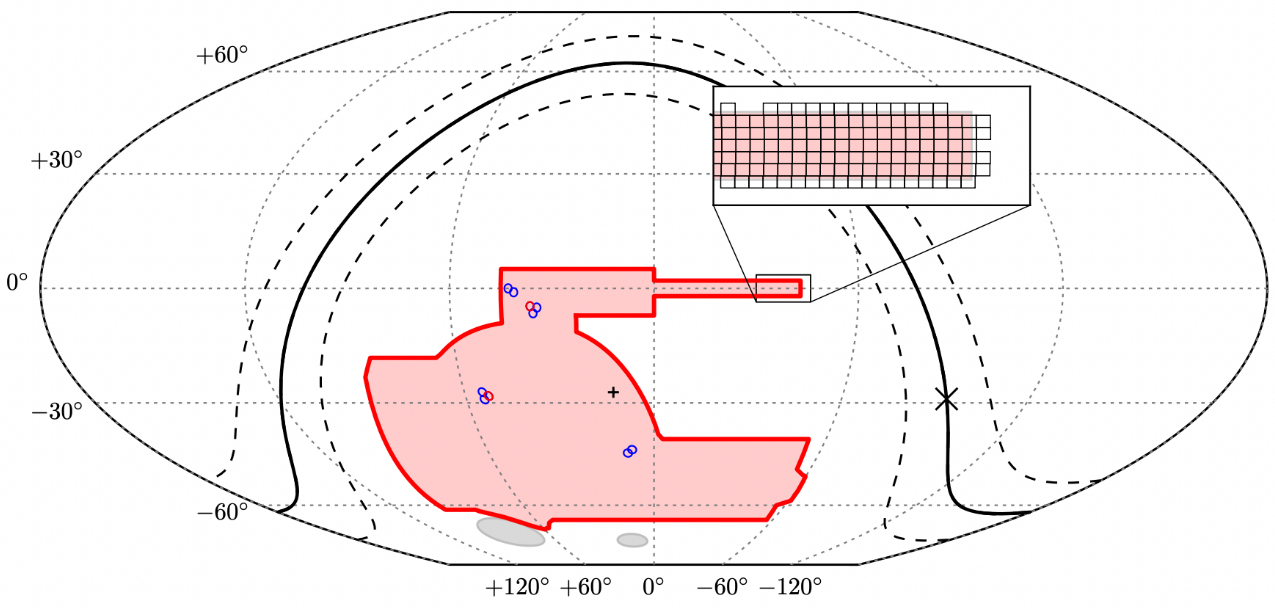

Figure 1.8: Figure of the DES survey area in celestial coordinates with the 5000 square degree footprint shown in red, taken from Abbott et al. (2018). The circles show supernova fields, and the plane of the Milky Way is given as the solid line.

et al., 2017), and in chapter 2 I use machine learning to show how the efficiency of searching for TNOs in DES data can be improved.

1.4 Machine Learning

Machine learning (ML) can be described as the general method of using computer programs which learn from example datasets how best to perform various tasks without explicit programming. This makes it an example of artificial intelligence (AI) which is generally thought of as being the broader topic, covering any intelligent, artificial system. While many tasks and analyses can be completed by programming algorithms with the exact steps required, and can still be deemed AI where an intelligent system is made by encoding domain knowledge, there are increasing areas where it can be too complex (or indeed impossible) to manually code an algorithm. Instead, effective algorithms can be made by implementing ML, where the ML algorithms are fine-tuned to each problem through the datasets used and the optimisation process.

The first recorded use of ML came from Samuel (1959) who used a neural network to learn how to play checkers better than a human. Initially the field of ML was seen as a useful form of pattern recognition and was deeply connected to the hopes of creating more powerful AI. It wasn't until later in the 20^{th} century when ML methods began to thrive having shifted away from the goal of creating an AI and instead moved towards applying the methods to problems across many different disciplines. Now, ML is used for almost any problem, with the largest areas including clustering for pattern recognition (Jain et al., 2000), classifications (Storrie-Lombardi et al., 1992), anomaly detection (Chandola et al., 2009), and regression analyses (Pasquet et al., 2019).

There are four main categories of ML which are dependent on how the algorithm learns and the data used as an input. The first is supervised learning which uses a labelled dataset where the desired output is already known. The algorithms then learn how to map features of the input data to the known output. Second is unsupervised learning which doesn't have labelled data. Instead, it learns to find structure and recognise patterns in the input data, and in some cases when combined with supervised methods in the third method of 'semi-supervised' learning, the combination of labelled data with additional unlabelled examples can result in an improved final model. Finally, there is reinforcement learning which uses algorithms that attempt to perform a task, and by providing feedback to the algorithms (rewarding when correct and penalising when wrong), they learn to maximise the rewards and thus improve accuracy.

In this thesis I describe in detail the supervised ML algorithms most commonly used in research, and which I applied to problems addressed in the following three chapters of classifying possible TNOs and providing effective photo-z estimates. As I was implementing ML for both classification and regression problems, the methods below discuss both with any differences highlighted.

1.4.1 Supervised Machine Learning Methods

1.4.1.1 Linear methods

Possibly the most simple example of a ML algorithm, linear regression aims to fit a linear model to the input data. This is done by making the target output a linear combination of the input features as described by

$$\hat{y}(w,x) = w_0 + w_1 x_1 + ... + w_N x_N. \tag{1.12}$$

For the target $\hat{y}$, and features x_i, the coefficients, w_i, are set which aim to minimise the squared residuals between the observations and the predictions akin to least squares regression. This means the loss function is simply $(\hat{y}(w,x) - y)^2$ and the objective which is minimised is just the average for all N samples:

$$\frac{1}{N} \sum_{i=1...N} (\hat{y}_i(w,x) - y_i)^2. \tag{1.13}$$

The squared residuals are chosen as the loss function rather than the absolute values to be able to have a smooth derivative. This is important for finding the optimal values for the coefficients which require a closed form solution for the derivative, and by setting the gradient $= 0$, the solution to the system of equations will then give the best values of w_i. An additional benefit to using the squared residuals is that it emphasises the differences between predictions and true values.

For classification rather than regression, despite its name, logistic regression is used. The misleading name comes from the fact that it uses the logistic function,

$$f(x) = \frac{1}{1+e^{-x}}. \tag{1.14}$$

As during classification there are only a set number of possible targets, $\hat{y}$, and in the case of binary classification only two possibilities, 0 (for a negative result) or 1 (for a positive result), the logistic function is used as it has a codomain of (0, 1). This is useful as when the model returns a value closer to 0, it classifies as being negative, and similarly if the model returns a value closer to 1, it classifies as positive.

Using the logistic function, the equation for the predictions becomes

$$\hat{y}_{(w,x)} = \frac{1}{1+e^{-(w_0+w_1x_1+\ldots+w_Nx_N)}}. \tag{1.15}$$

However, instead of attempting to minimise the loss function (the mean squared error (MSE)), in logistic regression the likelihood function is maximised. A likelihood function defines the probability of observing an example according to the model. This means that for a labelled example (x_i, y_i) in the training data, applying the model will give an output $0 < p < 1$. For a positive class y_i, the likelihood of y_i being classified as positive by the model is p, whereas if y_i is a negatively classed example then the likelihood of it being classed negative is $1-p$. The likehood function is given by

$$L = \prod_{i \ldots N} \hat{y}_{(w,x)}^{y_i} (1-\hat{y}_{(w,x)})^{1-y_i}. \tag{1.16}$$

But as the logistic function uses exponentials, it is more practical to maximize the log-likelihood which is given by

$$logL = \sum_{i=1}^{N} y_i ln(\hat{y}_{(w,x)}) + (1-y_i) ln(1-\hat{y}_{(w,x)}). \tag{1.17}$$

As there is not a closed form solution to the log-likelihood, a numerical optimisation procedure is required. There are many possible ‘solvers’ such as gradient descent, but all work by using iterative optimisers which aim to locate a local minimum of the function. They start at a random point and take steps which are proportional to the gradient at that point.

In both regression and classification the estimates of the coefficients rely on the assumption that the input data features are independent. If the features are correlated the output predictions become very sensitive to any errors present in the observations, and as the outputs can vary wildly, the resulting model has a large variance and can be prone to overfitting[1] (Hastie et al., 2009).

1.4.1.2 *k*-Nearest Neighbours

k-Nearest Neighbours (*k*NN) is an example of an instance-based (non-parametric) ML algorithm. This means that unlike other algorithms which, once trained, are able to make predictions on new data without using the training data, *k*NN retains all the training data in memory. It does this as to make a prediction for a previously unseen point, *k*NN finds a predefined number of points, *k*, in the training data that are closest to the unseen point and returns the majority classification (in the case of a classification problem) or average (in the case of a regression problem) as its prediction (Altman, 1992).

The measure of how close two points are is given by a distance function which could in theory be any metric measure, but the Euclidean distance is the standard choice. The Euclidean distance is given by

$$d(x,x') = \sqrt{\sum_{i...N}(x_i - x'_i)^2}, \tag{1.18}$$

which is simply the square root of the sum of the squared difference between points x and x'.

As there are many other measures of distance which could be chosen (such as Minkowski distance, Chebyshev distance, and Manhattan distance which are also commonly used), the distance measure is one of the 'hyperparameters' of the algo-

[1]ML models are said to be 'overfitting' when they perform very well during training and testing but fail to generalise to new datasets. This is a common issue in ML and efforts must be taken during the optimisation stages to ensure the models will give valid predictions for new data.

rithm. Hyperparameters are parameters which are set prior to the learning process in ML and dictate how the algorithms are built. In the case of *k*NN the value for *k* is another vital hyperparameter which dictates how many nearest neighbours are used to make the predictions for new points. I give more details about how hyperparameters are selected during the optimisation stage in section 1.4.3.

Although *k*NN is a relatively simple model, it has a highly successful track record and performs very well at various classification and regression problems including obtaining photo-*z* estimates (Ball et al., 2007). The instance-based *k*NN can often be more successful for problems with complex decision boundaries as decreasing the value of *k* makes the classification boundary more distinct and hence allows for classification of highly irregular data. However, by decreasing the value of *k*, one risks making the decision boundary too specific to the training data which results in overfitting. A larger value for *k* will suppress noise and make the classification more general, but could also result in the boundary not being distinct enough (this is displayed in figure 1.9 which gives an example of how changing the value of *k* effects the decision boundary). The optimisation process is vital to ensure an effective model is generated which also generalises well to new data.

1.4.1.3 Naive Bayes

Naive Bayes is a supervised machine learning algorithm that applies Bayes theorem with the ‘naive’ assumption that each pair of features is independent (Hand & Yu, 2001). For the class variable y, and a dependent feature vector x_i, Bayes theorem can be applied, where maximum a posteriori estimation is used to obtain an estimate for $P(x_i|y)$ and $P(y)$, where $P(y)$ is simply the relative frequency of class y in the training set.

Typically, a ’Gaussian Naive Bayes’ algorithm is used where the likelihood is taken to be Gaussian and the parameters μ_y and σ_y are estimated using maximum likelihood estimation. Then, using the probabilities given by

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right), \tag{1.19}$$

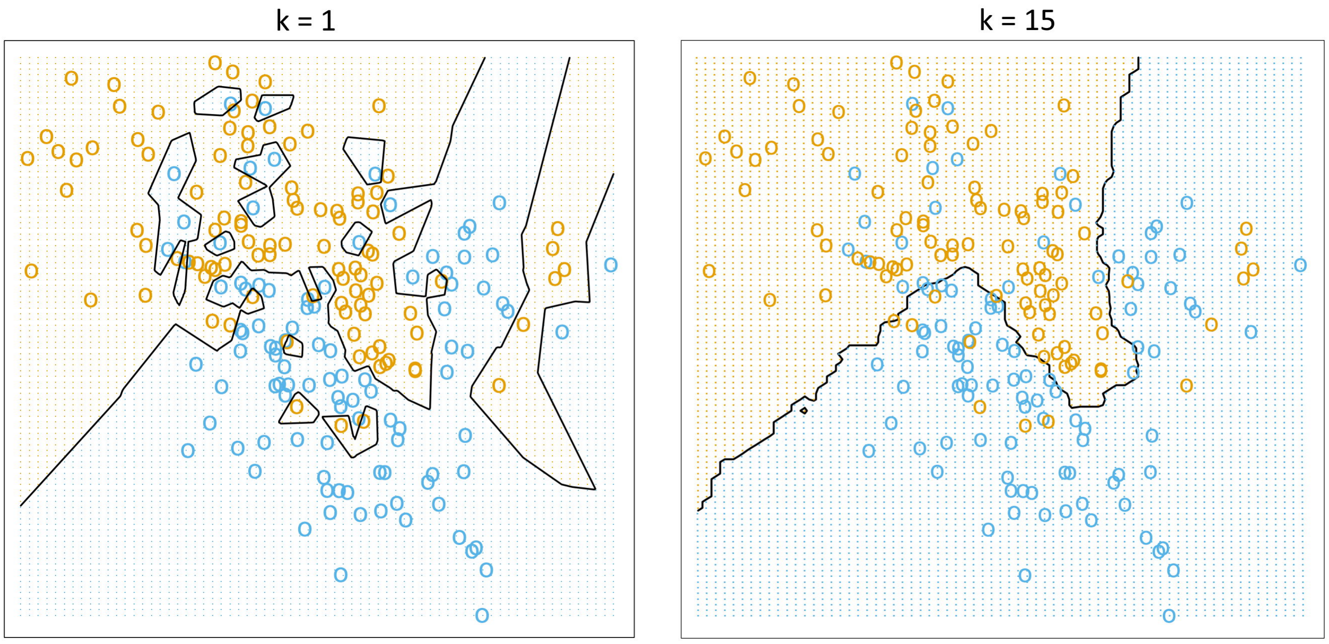

Figure 1.9: Figure taken from Hastie et al. (2009) showing how a k-Nearest Neighbours algorithm changes depending on the value of k. For smaller k values the decision boundary between the two different classes, shown here in orange and blue, is seen to be very specific to the training data, whereas for the higher value of k the boundary is smoother and more likely to generalise to new data.

Bayes theorem is applied to obtain the final probability of obtaining the class y given the features X which gives the classification result:

$$P(y \mid X) = P(x_1 \mid y) \times P(x_2 \mid y) \times \cdots \times P(x_i \mid y) \times P(y). \tag{1.20}$$

1.4.1.4 Decision Trees

Decision Trees are another example of non-parametric algorithms which use branching graphs built from simple decision rules to make predictions. At each branching node, a feature of the input data is inspected and a basic if-then-else statement is used to split the data depending on whether the value is above or below a certain threshold (Breiman et al., 1984). After some number of splits set by the depth of the tree, a leaf node is the reached which gives the final decision or prediction.

There are many different decision tree algorithms which change the ways in which the trees learn and grow. In this thesis I focus on Iterative Dichotomiser 3 (ID3) which was created by Quinlan (1986) and subsequent models which have become the most widely used. ID3 works by starting with the first 'root' node at the top of the tree and working down, and at each node it searches for the feature which results in the largest information gain (the biggest difference) for each split. Mathematically it does this using the following average log-likelihood as the optimisation criterion

$$\frac{1}{N}\sum_{i=1}^{N} y_i ln f_{ID3}(x_i) + (1-y_i) ln(1 - f_{ID3}(x_i)), \tag{1.21}$$

where there are N labelled examples with feature values x_i and label y_i, and f_{ID3} is the decision tree.

If we call S the set of labelled training data, then for the first root node which still contains all the data we have $S = \{(x_i, y_i)\}_{i=1}^{N}$. The model can then be thought of as being constant where it would always give the same output, y, for the single node with

$$f_{ID3}^{S} = \frac{1}{|S|}\sum_{(x,y)\in S} y. \tag{1.22}$$

As there are multiple features, j, which make up the feature matrix, x_i^j, the different features are searched at set thresholds, t, to split the set S into two subsets: $S_- = \{(x,y)|(x,y) \in S, x^{(j)} < t\}$, and $S_+ = \{(x,y)|(x,y) \in S, x^{(j)} \geq t\}$. For all possible combinations of (j,t) the splits of (S_-, S_+) are evaluated to find the split which results in the largest information gain and minimises the entropy. The entropy, H, measures uncertainty, being highest when all variables have an equal probably and being minimised when the variables can only have a single value, and for a set S, the entropy has a value

$$H(S) = -f_{ID3}^{S} ln f_{ID3}^{S} - (1 - f_{ID3}^{S}) ln(1 - f_{ID3}^{S}). \tag{1.23}$$

The entropy of the split, $H(S_-, S_+)$, is then simply the weighted sum of the two entropies given by

$$H(S_-, S_+) = \frac{|S_-|}{|S|}H(S_-) + \frac{|S_+|}{|S|}H(S_+), \tag{1.24}$$

and at each stage of ID3, to split the data, the splits are found which minimise the entropy.

The algorithm ends at leaf nodes where no more splits happen for the following reasons. First, if all examples in the node are correctly classified then there is no change from splitting the data. Second, if there is no feature which will helpfully split the data or if the split would only reduce the entropy by a value less than a set threshold (which is set during optimisation). Finally, if the tree reaches its maximum depth (a hyperparameter which is also set prior to the learning process) then the tree stops branching and the last nodes are taken as leaf nodes.

Other tree algorithms have been made which aim to improve upon ID3 in various ways. C4.5 was introduced by Quinlan (1993) as ID3's successor and allowed for continuous variables to be used instead of requiring features to be categorical. This was done by using bins which partitioned the continuous variable into discrete sets of intervals. C4.5's other major improvement was the ability to order the splits. ID3 was limited by only considering the best split at each local node, and as it works from top to bottom, by not considering the future splits the algorithm doesn't necessarily find the optimal solution. C4.5 uses backtracking whereby the trained tree is converted into if-then-else statements and by evaluating the accuracy of each statement, the order in which they should be applied can be determined.

Another successful algorithm is Classification and Regression Trees (CART) (Breiman et al., 1984). CART performs very similarly to C4.5, also growing the trees by using the feature splits with thresholds that give the largest information gain and recursively selecting the best splits, but CART also supports numerical results allowing it to be used for regression problems as well as classification. As such CART was the algorithm used in the following chapters.

Decision trees can be very effective ML algorithms for a variety of problems and often yield high accuracies, however, they can also quickly become complex and are prone to being overfitted to the training data. There are ways that decision trees can be made more robust to better generalise, such as ‘pruning’ where less relevant nodes are replaced with leaf nodes to reduce the tree size and model complexity, but perhaps the most common solution is to use ensemble methods (such as boosted decision trees, random forests, and extremely randomised trees) which combine multiple decision trees.

1.4.1.5 Boosted Decision Trees

Boosted decision trees are an example of a ML algorithm which makes use of ‘boosting’. Boosting methods sequentially combine multiple weaker learners into a single strong learner with the aim of creating an ensemble method with reduced bias (and an improved accuracy) (Kearns & Valiant, 1994). There are different types of boosted trees with the most popular algorithms being Adaboost (Freund & Schapire, 1997), and the more general gradient boosted decision trees (Friedman, 2002).

Both methods work by starting at the same point as ID3, with a single model defined by

$$f = \frac{1}{N}\sum_{i=1}^{N} y_i. \tag{1.25}$$

The training data is then modified, and instead of using the label, y_i, the residuals, $\hat{y}_i = y_i - f(x_i)$, can be calculated and used to train a new tree, f_1. The boosted model is then defined as

$$f = f_0 + \alpha f_1, \tag{1.26}$$

where α is the learning rate (which is a hyperparameter that gets set during optimisation). The residuals of this model can be recalculated and used to train the next tree, f_2, and so on until a maximum number of trees, n, is reached. The residuals suggest how good the predictions made by the model are, hence by training a new model to reduce the residuals and adding it to the existing model, each tree in the ensemble acts to reduce the errors of the previous trees.

As the boosted trees are built forward they can also be written as

$$F_m(x) = F_{m-1}(x) + h_m(x), \tag{1.27}$$

where the model $F_m(x)$ is given in terms of the previous model, $F_{(m-1)}$, and the newest added tree $h_m(x)$. This is useful as it allows us to also write the crucial stage in the machine learning process, where the tree $h_m(x)$ is fitted to minimise the loss function, l.

$$h_m = \arg\min_h L_m = \arg\min_h \sum_{i=1}^{n} l(y_i, F_{m-1}(x_i) + h(x_i)), \tag{1.28}$$

and using the Taylor expansion, the value of the loss, l, is approximated as:

$$l(y_i, F_{m-1}(x_i) + h_m(x_i)) \approx l(y_i, F_{m-1}(x_i)) + h_m(x_i) \left[\frac{\partial l(y_i, F(x_i))}{\partial F(x_i)} \right]_{F=F_{m-1}}. \tag{1.29}$$

Setting g_i equal to the derivative of the loss (the gradient),

$$g_i = \left[\frac{\partial l(y_i, F(x_i))}{\partial F(x_i)} \right]_{F=F_{m-1}}, \tag{1.30}$$

then allows the tree to be written as

$$h_m \approx \arg\min_h \sum_{i=1}^{n} h(x_i) g_i. \tag{1.31}$$

This then minimises when, $h(x_i)$ is proportional to $-g_i$. So each tree is fitted to estimate the negative gradient of the data, hence the name gradient boosting. Adaboost is a specific case of a gradient boosted decision tree, where the loss function is the exponential loss.

1.4.1.6 Random Forests

Random forests were first introduced by Ho (1995) as an example of applying ‘bagging’ (a shortening of bootstrap aggregating) to decision trees. Rather than building an ensemble from many sequential estimators like in boosting, bagging uses many independently built estimators and averages their outputs to give a final prediction

(Breiman, 1996). By taking the average, these methods typically improve upon the single weak learners by drastically lowering the variance, and hence also act to reduce the chance of overfitting.

A random forest, similar to a boosted decision tree, also uses many single trees, however, it differs greatly in how the trees are grown and combined. The process begins by using random samples of the training data to train each individual tree. These samples are made by taking the overall training set and sampling with replacement (allowing repeat observations to occur in the sample); this means that the random samples made are independent. If n total decision trees are grown then the prediction is simply the average of the n predictions:

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x). \tag{1.32}$$

There is an additional step of randomness which separates random forests from simply being standard bagging. Rather than using normal decision trees which branch based of taking the best possible split and results in the largest information gain, the trees in the random forest take the best split from a random subset of the features. While this yields trees which don't always have the best splits, it ensures that the trees are less correlated. In the event of only a few features having the greatest importance, they would always be more likely to be chosen to split the data, and hence without this extra step of using a random subset of features to split the data the trees would all be correlated and their averaged prediction wouldn't be as great an improvement.

As mentioned, the overall effect of averaging the predictions of the different trees in the forest is an ensemble with a significantly reduced variance. Furthermore, the errors of the individually worse performing trees in the forest in effect cancel out and result in random forest algorithms usually outperforming single decision trees in accuracy as well as being a more robust, and therefore useful, model.

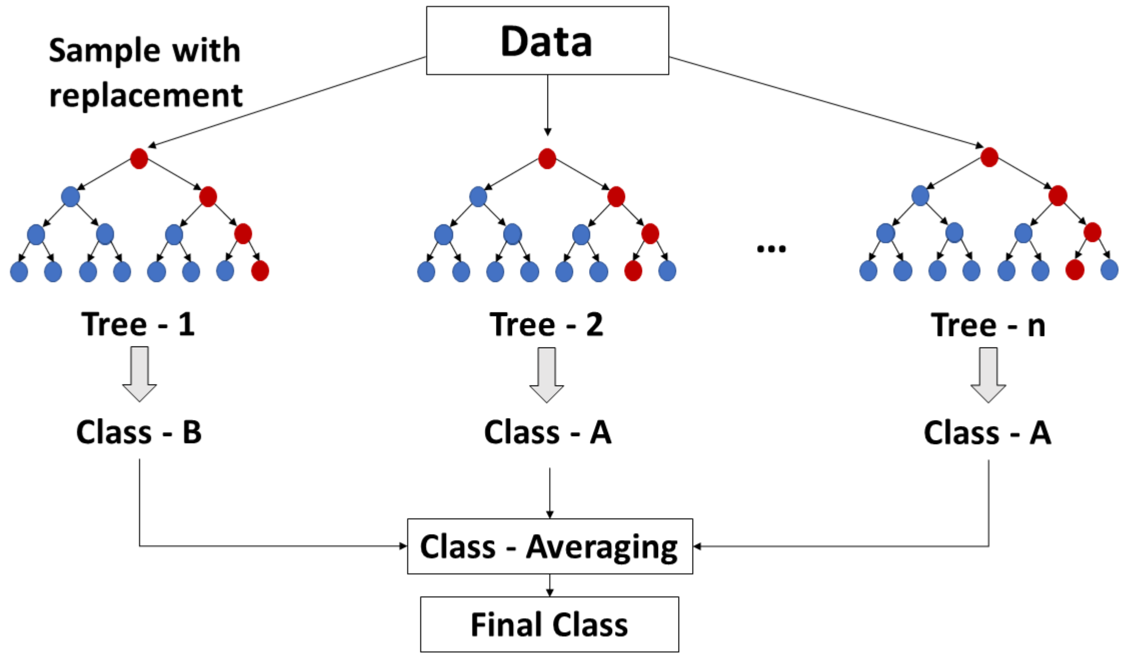

Figure 1.10: Figure from Henghes et al. (2020) showing the construction of a random forest classifier. The data is sampled with replacement resulting in many decision trees being trained on random subsets of the data. A second element of randomness is then added where the feature splits performed in each tree are no longer the splits resulting in the highest information gain, but rather the best splits are taken from a random subset of the features. The class outputs of all the individual trees in the forest are averaged to give the final prediction, which has a much reduced variance and isn't as prone to overfitting as a single decision tree.

1.4.1.7 Extremely Randomised Trees

Extremely randomised trees (Geurts et al., 2006) is another ensemble variation of decision trees. They are very similar to random forests, but with a difference which adds an additional stage of randomness to try to further reduce the variance.

This extra random step works by using random thresholds for the feature splits. By selecting the best threshold out of a set of random thresholds instead of the threshold which results in the highest information gain, the resulting trees have a third random aspect (on top of the random subsets of data and random features that splits are performed on). This acts to further reduce the correlation between trees that make up the forest and ensures that the performance of the ensemble improves. The predominant reason for the improved performance is that the predictions of

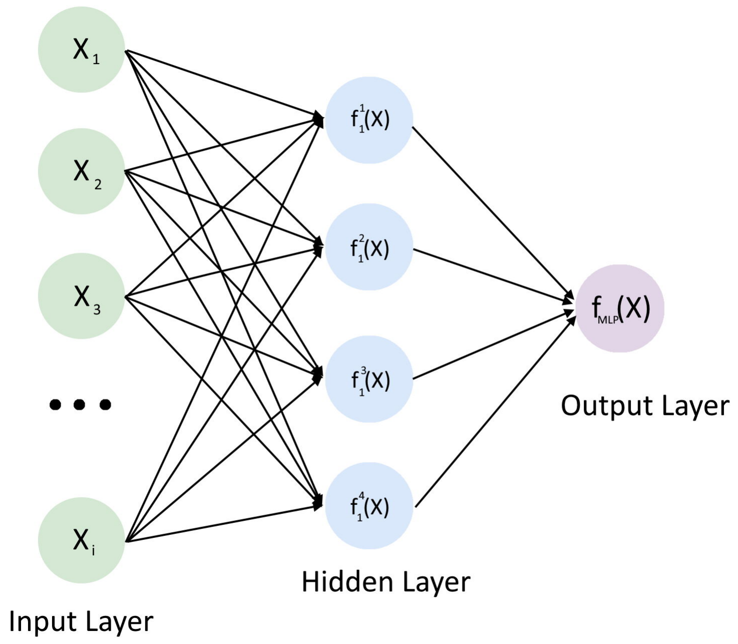

Figure 1.11: Figure showing the layout of a simple multi-layer perceptron with one hidden layer with four nodes. Each node is shown as a circle and is fully connected to each node in the previous and following layers.

more accurate trees will likely agree whereas worse trees will disagree, however, if the trees are correlated then the bad tree are more likely to also agree which adds bias. Hence by ensuring that the trees in the forest are independent and not correlated, the averaging of the trees to give the final prediction will yield an improved model.

1.4.1.8 Multi-layer Perceptrons

A multi-layer perceptron is an example of a fully connected neural network made with at least three layers of nodes. Consisting of an input node, an output node, and a minimum of one hidden layer, multi-layer perceptrons are the most simple examples of deep neural networks and can be visualised in figure 1.11. Each node is shown as a circle with the arrows showing the connections and inputs / outputs of the nodes. In fully connected networks such as this, each of the layers receive the output of the previous layer as their input.

The input layer contains nodes which represent each individual feature, then in each node of the hidden layers the process is as follows. First, the inputs of each node in the previous layer are joined to give the input vector fed into the node. Each node can then be thought of as a linear transformation, acting similarly to logistic regression to transform the input. However, the network differs from logistic regression as the combination of many nodes with different weighted transformations allows it to approximate any continuous function. The linear transformation is followed by applying an activation function to obtain the output. At the end of the network, an output layer is used which typically has a single node (unless multi-class outputs are required). If this final node uses a linear activation function, the final output is continuous (e.g. for regression problems); alternatively a logistic activation function can be used for classifications.

Mathematically we can write the overall function of the network, f_{MLP}, as a combination of the individual layers. So for a three layer network it could be written

$$y = f_{\mathrm{MLP}}(x) = f_3(f_2(f_1(x))), \tag{1.33}$$

where f_l are vector functions with the form

$$f_l(x) = g_l(\underline{\underline{W_i}}x + b_i). \tag{1.34}$$

In the above equation, g_l is the activation function which is a fixed function chosen beforehand, and the parameters $\underline{\underline{W_i}}$ (a matrix) and b_i (a vector) are learned during the training through gradient descent with a chosen cost function (typically the mean squared error). These parameters can therefore change between nodes in the same layer and are updated using backpropagation which is an efficient method for computing the gradient (Rumelhart et al., 1986). In backpropagation the gradients are calculated from the final layer backwards, reusing the previous layers' computations to speed up the calculation of the gradient for the earlier layers.

Gradient descent updates the parameters at each iteration of the training where their change is proportional to the derivative of the cost function. A common problem which can occur with gradient descent is vanishing gradients, which slows down training and in extreme cases can stop neural networks from training altogether. Many activation functions (e.g. the hyperbolic tangent) have gradients in the range $(0,1)$. As backpropagation multiplies these gradients using the chain rule, the multiplying of the small gradients decreases exponentially. This has the effect of massively slowing the training of the earlier layers, however, there have been modern implementations which aim to fix this issue with the rectified linear unit (ReLU) activation function given below being the most popular (Nair & Hinton, 2010).

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{1.35}$$

1.4.1.9 Convolutional Neural Networks

Deep neural networks such as multi-layer perceptrons with many hidden layers very quickly grow in size. To be exact, adding an additional hidden layer of size S_l results in a further $S_l(S_{l-1}+1)$ parameters (where S_{l-1} is the number of nodes in the previous layer). This quickly leads to models with billions of parameters which can make optimisation too computationally intensive to allow for a useful model. This problem is especially evident for tasks where the input has many dimensions such as when the training data is made up of images.

Convolutional neural networks (CNNs) are specially designed neural networks which were made with the purpose of reducing the number of parameters and hence allow deeper networks to be trained. Similar to other artificial neural networks such as the multi-layer perceptron, CNNs aim to mimic the processes in the human brain (McCulloch & Pitts, 1943). They are made of many interconnected nodes (or neurons) which are used to process information similar to the biological counterparts that inspired them. CNNs were inspired by the visual cortex where the neurons only respond to stimuli in specific regions called receptive fields. The receptive fields of neurons overlap to cover the entire field of view (Hubel & Wiesel, 1968). CNNs, as their name suggests, use convolutional layers which act to convolve the inputs in a way that mimics the visual cortex.

CNNs are the leading ML algorithm when it comes to using images as an input.

They work particularly well for images as the pixels which are close together in an image usually include similar information. The exception comes at edges, where parts of an image with different objects (such as galaxies and the background sky) meet. If a network can be trained to recognise these edges and similar regions then they can make informed predictions about the images. As the most vital information is therefore local within the images, the images are inspected in patches with a moving window to cover the entire region. Many smaller regression models can then be trained for each patch with the goal of learning specific patterns within the small region.

This is done using filters (also called kernels) which are matrices with the same dimensions as the patches of the image. The filters of each layer (of which there can be many, similar to nodes in standard networks) are applied in a sliding motion across the volume of the input data, left to right and top to bottom, as shown in figure 1.12. The model computes the dot product between the filter values and input values, as well as adding a bias, to give the convolved features which then get passed through an activation function to produce an activation map (also called a feature map).

The values of the filters and bias are initially random and get updated during the training as the model learns which parameters produce more relevant values. This is done in the exact same way as other networks, through gradient descent with backpropagation to minimise a given cost function. The activation function used on the convolved features allows for nonlinearity to be added, with the most popular function being ReLU (which was defined in equation 1.35). Finally, the feature maps from all filters are stacked to give the output of the convolutional layer.

CNNs typically have multiple convolutional layers following one another, with the subsequent layers taking the output of the previous layer (the stacked feature maps) as their input. These collections of matrices (called volumes) then have the filters of the new layer applied, where the convolution of a volume is simply a sum over each filter as shown in figure 1.13. As CNNs take images as the original input, these are often volumes too as the images are typically represented by multiple

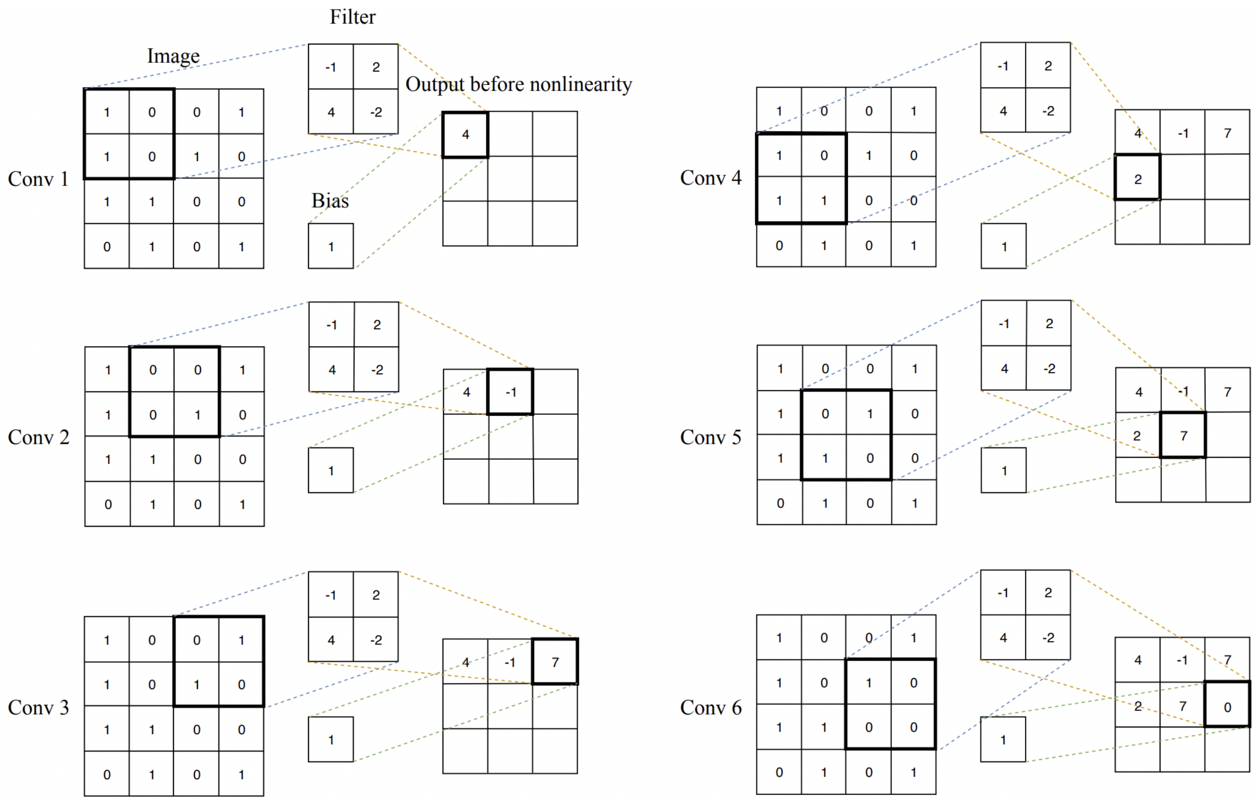

Figure 1.12: Figure from Burkov (2019) displaying how CNN convolutional layers work by applying a filter to the image in a sliding motion (here with a stride equal to one).

channels (e.g. RGB for colour, or ugriz filters as in chapter 4). The additional layers further convolve the original input, reducing the number of overall parameters and allowing for deeper networks to be constructed.

Two important hyperparameters of CNNs are the stride and padding of the filters. The stride dictates how the filter moves across the image, so with a stride equal to one, the filter moves across one pixel at a time. By increasing the stride one can reduce the spatial dimensions of the output volumes and hence it is another method of reducing the computational resources required by the model. The padding dictates how the model treats the edges of the convolution where, if the stride is such that the filter doesn't fit perfectly inside the image, the image will be padded with zeros or the strip of the image where the filter doesn't fit is ignored.

There are two more types of layers which along with the convolutional layers make up a CNN. First, pooling layers are layers which quickly reduce the volumes

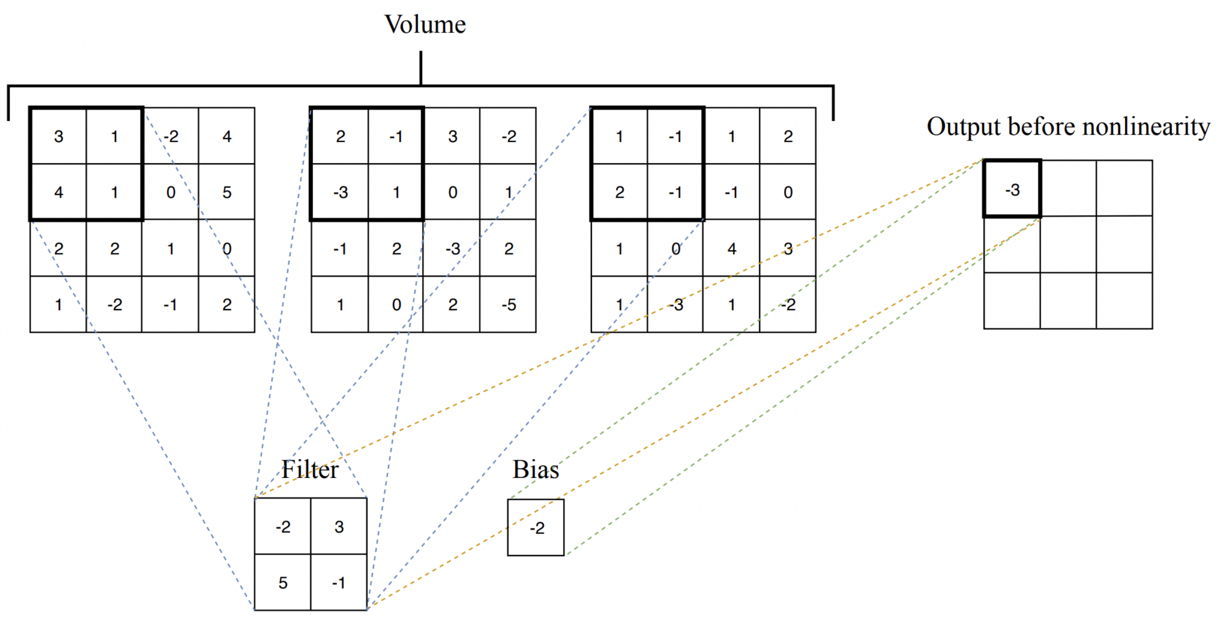

Figure 1.13: Figure also from Burkov (2019) displaying how filters are applied to volumes in convolutional layers.

and number of parameters in the network. There are three different types of pooling: max pooling, average pooling, and sum pooling. Each works in the same way and an example is given in figure 1.14. A ‘filter’ of a set size (although different to convolutional filters as they don’t contain any values themselves) is applied to the input volume in the same vein as the convolution layer, in a moving window with a set stride. As the names suggest, in max pooling the operation is simply taking the maximum value, average pooling takes the average, and sum pooling takes the sum of the elements in the filter.

The other type of layers used in CNNs are fully-connected layers. These work by first flattening the output volume of the convolutional network into a single feature vector. This can then be used as the input to the fully connected layers which when put together are the same as a standard fully connected neural network as described in section 1.4.1.8. These fully connected layers act to take the transformed input, where the convolutional layers have acted to extract the features of the images, and process the features before using a final layer with an activation function to give the desired output.

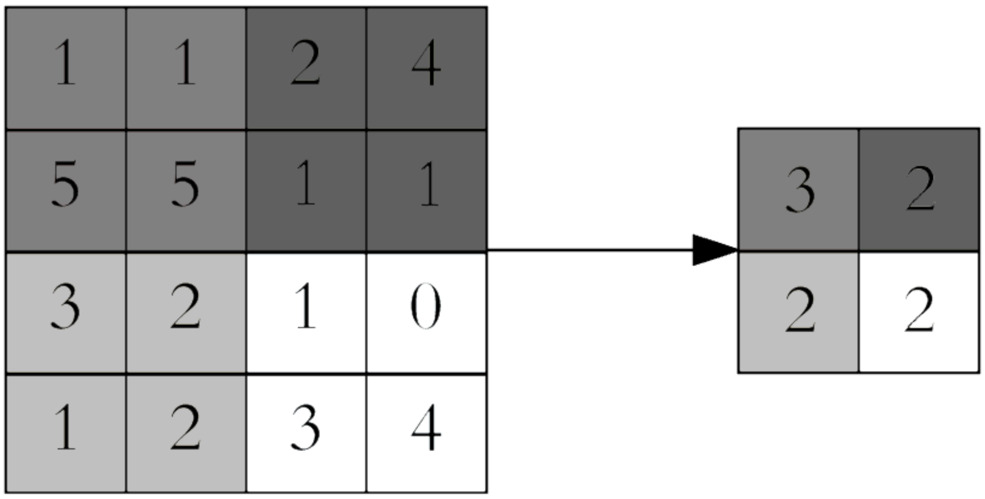

Figure 1.14: Figure showing how pooling layers are applied to reduce the size of the input volume. In this example average pooling is used with a filter size of (2×2) and a stride equal to two.

1.4.1.10 Inception module networks

Inception module CNNs are a specific type of CNN which make use of inception modules. These modules have the structure shown in figure 1.15 and were first used by Szegedy et al. (2015) to allow for deeper network architectures in the making of 'GoogLeNet'. They work in exactly the same way as normal CNNs, with the addition of these inception modules instead of a single convolutional layer.

The inception module works using sets of convolutional layers which are applied in parallel instead of the sequential convolutions of standard CNNs. The convolutional layers begin with a kernel size of (1×1) before then using convolutional layers with larger kernel sizes to allow for more efficient computations of the matrix dot product. This allows for deeper networks to be constructed without requiring as many computational resources, and which have been shown to improve performance (Szegedy et al., 2016).

1.4.2 Metrics

The metrics one uses are vital not just in evaluating the performance of machine learning models but also while building the models. Before any decision can be made for which hyperparameters to set in the models, one first needs to decide which metric to optimise for. There are different metrics for regression problems,

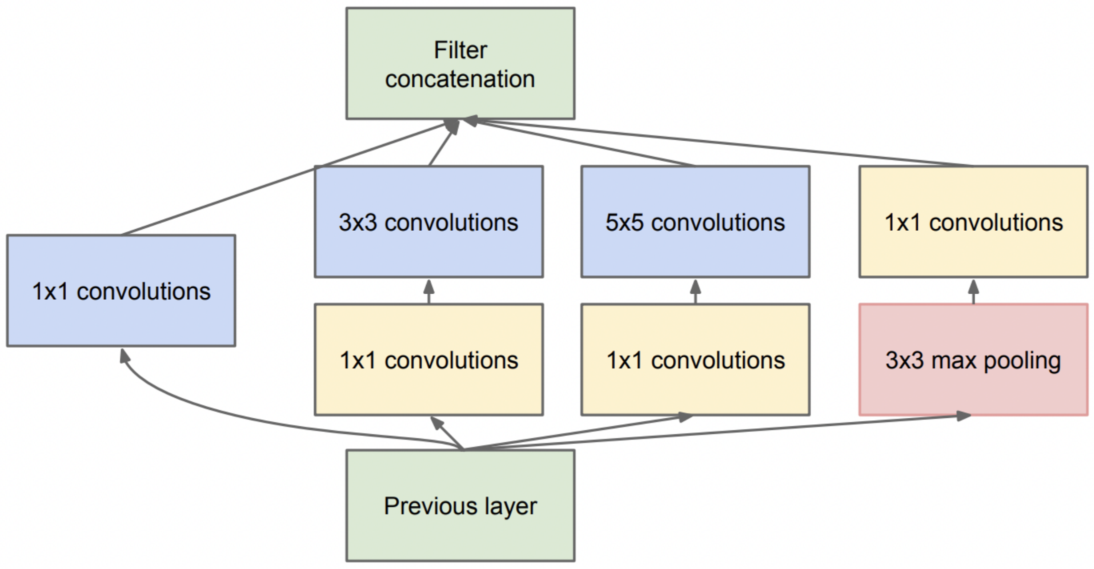

Figure 1.15: Figure from Szegedy et al. (2015) showing the structure of a typical inception module.

which can simply use errors based on the residuals of the predictions, compared to classification problems, where generally the metrics will be derived from a confusion matrix.

1.4.2.1 Regression metrics

There are three main error metrics used for regression problems. The most widely used is the mean squared error (MSE), defined as

$$\mathrm{MSE}(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{1.36}$$

which could also be square rooted to give the root-mean squared error (RMSE); second is the mean absolute error (MAE) given by

$$\mathrm{MAE}(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|; \tag{1.37}$$

and lastly is the R squared score (R^2) defined as

$$R^2(y,\hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}. \tag{1.38}$$

The equations for each of these metrics is given above where for the i-th example within a total of n examples, $\hat{y}_i$ is the predicted value, y_i is the true value, and $\bar{y}$ is the mean of y_i.

While these general error metrics are widely used for various problems, any error metric could be used and typically more problem-specific metrics will also be calculated depending on the application of the machine learning model.

1.4.2.2 Classification metrics

As mentioned, classification metrics are typically derived from the confusion matrix. This is a matrix of values that summarises the performance of the machine learning models by showing the number of examples in each of the different classes for both the predicted and true datasets. In table 1.3 I give an example of a confusion matrix for a binary classification, where there are only two classes, however, for multi-label classifications there could be any number of different classes.

For this binary example there are four values given by the confusion matrix: True positives (TP), which are the number of examples correctly classified as positive; false positives (FP), which are the number of examples incorrectly classified as positive (so the true value would have been negative); true negatives (TN), which are the number of examples correctly classified as negative; and false negatives (FN), which are the number of examples incorrectly classified as negative (and which should have been positive).

The simplest metric for classification is the accuracy. The accuracy is just the fraction of the time that the classification is correct, and is given by the number of true examples over the total number of examples:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}. \tag{1.39}$$

While this is a very powerful metric which can be useful to quickly determine model performance, it can also be deceptive for classifications where there is a class imbalance. As explained in chapter 2 when searching for TNOs there were far fewer positive examples of true detections and the dataset was heavily imbalanced with

Table 1.3: An example confusion matrix for a binary classification problem where there are only two possible classes. This results in four values: The number of examples correctly classified as positive, true positives (TP); the number of examples incorrectly classified as positive (so the true value would have been negative), false positives (FP); the number of examples correctly classified as negative, true negatives (TN); and the number of examples incorrectly classified as negative (which should have been positive), false negatives (FN). These values can then be used to calculate various classification metrics.

		Predicted Class	
		Positive	Negative
True Class	Positive	TP	FN
	Negative	FP	TN

many more negative examples of false detections due to noise. This meant that a classifier could have a null accuracy (accuracy of predicting all examples to be the same class) which was still very high.

The accuracy is also only useful when used for problems where the errors of the predictions are equally important (i.e. we care equally about FP and FN values). Again using the example of chapter 2, it was far more important to minimise the number of missed detections (so the number of false negatives). This was because the model was being used to reduce the amount of detections due to noise being passed through the pipeline, but a model which also reduced the number of possible real objects wouldn't have been useful in a detection pipeline. As such there are other metrics which can be more informative about the numbers of FN or FP.

The recall (also known as sensitivity) and precision are two of most commonly used metrics for classifications. The recall gives the ratio of the TP to the total number of positive examples in the dataset, and the precision gives the ratio of the TP to the total number of positive predictions:

$$\text{Recall} = \frac{TP}{TP+FN}, \tag{1.40}$$

$$\text{Precision} = \frac{TP}{TP+FP}. \tag{1.41}$$

These metrics give more specific information than the accuracy with the recall describing how sensitive the model is (or how many positive examples are being missed by the classifier), and the precision describing the purity of the model and how much noise there is in the predictions (how many negative examples were classified as positive). An additional metric, the F1 score, can be made which is a combination the recall and precision and is defined as

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{1.42}$$

Finally, it can often be helpful to use plots to visualise the machine learning algorithms' performances. The receiver operating characteristic (ROC) curve is commonly used which plots the true positive rate against the false positive rate. The true positive rate is defined as the proportion of correctly predicted positive examples (making it the same as the recall) and the false positive rate is the proportion of incorrectly predicted negative examples:

$$TPR = \frac{TP}{TP+FN} = \text{Recall}, \tag{1.43}$$

$$FPR = \frac{FP}{FP+TN}. \tag{1.44}$$

By plotting the TPR against the FPR for many different decision thresholds (the threshold at which the model predictions change from negative to positive e.g. for a binary classification of (0, 1) the default threshold would be 0.5), one obtains a graph where the area under the curve (AUC) gives another good measure of model performance with the ideal model having an AUC = 1.

1.4.3 Optimisation

Optimisation is the process of fine-tuning the hyperparameters of machine learning algorithms which dictate how the algorithms are constructed and learn. Before performing optimisation a metric is chosen to optimise for (aiming to minimise or maximise the metric). For traditional algorithms there are three optimisation methods which can be used to select the optimal hyperparameters. The first and most complete method is brute force optimisation. In brute force optimisation every possible combination of a specified grid of hyperparameters is tested. This exhaustive method is guaranteed to give the best possible algorithm (the algorithm which minimises/maximises the chosen metric), however, it is very computational intensive and for large models and datasets can be impractical.

Instead, one of the other two optimisation methods can be used. Random optimisation takes the same grid of hyperparameters, and instead of testing every possible combination, it tests a set number of random combinations. With a high enough number of random sets of hyperparameters, random optimisation finds a good approximation of the best possible hyperparameters and is much faster than the brute force method. The final method is Bayesian optimisation. Bayesian optimisation (which is fully described by Snoek et al. (2012)) aims to iteratively test combinations of hyperparameters until it converges on the best result. In theory this method can give the best combination of hyperparameters, although in practice they are typically not an exact match to the brute force method. Bayesian optimisation does however remove the element of luck from random optimisation (which could yield both good or bad results), and is a good middle ground still being faster than brute force optimisation.

Deep learning algorithms such as CNNs cannot be optimised in the same way. While different numbers of layers and neurons can easily be specified, in practice it is very difficult to iteratively test all the different hyperparameters of the network (such as the kernel sizes, stride, padding) at the same time. Indeed, even when it is possible, the algorithms are much slower to train and it isn't feasible to carry out the same kind of optimisation. Instead, the process of finding the best network architecture is often more of a process of trial and error. By using domain knowledge and previous examples, the standard practice is to adapt previously existing networks and test different architectures to find a suitable model.

While optimising, another important process is cross-validation. Cross-validation works by taking the training data and splitting it into *k*-folds, which are different subsets of the same size. Using the example of 5-fold cross-validation, the training data is split into five folds, where each subset would contain 20% of the training data. When training the model, four of the five folds are used to train the algorithm which allows for the fifth fold to be used as a validation set. By iteratively varying the folds used in the training set, one obtains five different outputs, and these can be averaged to give the final value of the metric. This is far more robust than taking a single value as it helps prevent against overfitting a model to the training data (where it could perform brilliantly for one set of folds but not the others).

1.5 Benchmarking

Benchmarking is the practice of studying how specific models perform. This performance is described not just by the results of the various metrics described above, but also by the computational performance and efficiency. Conventionally benchmarking has been used to detail how a single architecture performs for a set problem, such as the LINKPACK challenge (Dongarra et al., 1979), however, in this thesis I was interested in a broader application to understand how various algorithms compared.

While machine learning has become an incredibly popular tool and is now used in almost all aspects of research, few investigations have been done to study the efficiency of the methods implemented. In chapters 3 & 4 I compare algorithms using various error metrics as well as investigating their scaling and speeds. This was done by recording the system information, which is described by the time, CPU usage, memory usage, and disk I/O, throughout the training and testing of the machine learning algorithms.

One of the main challenges of benchmarking is determining which metrics or figures are best used to outline the performance. This will be dependent on the type of problem being investigated as well as the purpose of the benchmarking. In chapters 3 & 4 I was predominantly interested in showing the scaling of the algorithms with different numbers of galaxies included in the training sets, and the effect on the error. As such, the plots comparing the number of points in the training set with the times taken during training or testing, and also comparing these with the MSE, were the plots chosen to highlight the performance.

Chapter 2

Searching for Trans-Neptunian Objects in the Dark Energy Survey using Machine Learning

This chapter is based on Henghes et al. (2020): Machine Learning for Searching the Dark Energy Survey for Trans-Neptunian Objects, Publications of the Astronomical Society of the Pacific (PASP) 133 014501.

In this chapter we investigate how implementing machine learning could improve the efficiency of the search for Trans-Neptunian Objects (TNOs) within Dark Energy Survey (DES) data when used alongside orbit fitting. The discovery of multiple TNOs that appear to show a similarity in their orbital parameters has led to the suggestion that one or more undetected planets, an as yet undiscovered 'Planet 9', may be present in the outer Solar System. DES is well placed to detect such a planet and has already been used to discover many other TNOs. Here, we perform tests on eight different supervised machine learning algorithms, using a dataset consisting of simulated TNOs buried within real DES noise data. We found that the best performing classifier was the Random Forest which, when optimised, performed well at detecting the rare objects. We achieve an area under the receiver operating characteristic (ROC) curve, $(AUC) = 0.996 \pm 0.001$. After optimizing the decision threshold of the Random Forest, we achieve a recall of 0.96 while maintaining a precision of 0.80. Finally, by using the optimized classifier to pre-select objects, we are able to run the orbit-fitting stage of our detection pipeline five times faster.

2.1 Introduction

The idea that additional planets may be present in the outer solar system has existed in astronomers' minds since the successful prediction and subsequent discovery of Neptune in 1846 (Le Verrier, 1839, Galle, 1846). Indeed, the discovery of the once major planet Pluto came as a direct result of a rush to find further planets (Tombaugh, 1946). After finding many other minor bodies in the outer solar system, the possibility of there still being a large planet left to discover seemed unlikely. However, recent detections of more Trans-Neptunian Objects (TNOs) have led to a resurgence in hunting for the elusive 'Planet 9'.

This rekindled excitement was caused by an observed similarity of orbital parameters of certain TNOs, and was first noted by Trujillo & Sheppard (2014) in their detection of $2012VP_{113}$. Objects like $2012VP_{113}$ have higher eccentricities, inclinations, and orbit further from the sun than the majority of TNOs, giving them the name 'Extreme-TNOs'(ETNOs). ETNOs typically have a semi-major axis, $a > 150$ AU, and a perihelion distance, $q > 30$ AU, and it was shown that these objects displayed a grouping with similar arguments of perihelia, $\omega \approx 0^\circ$, that could be explained by a large distant planet. The initial theory was that this planet caused the similar orbital elements via the Kozai mechanism (Kozai, 1962) whereby the oscillating argument of perihelia of the objects, about a value of either $\omega = 0^\circ$ or $\omega = 180^\circ$, would cause an exchange between the eccentricity and inclination of the body. However, this seemed improbable due to the lack of observations of TNOs with $\omega = 180^\circ$ (de la Fuente Marcos & de la Fuente Marcos, 2014). Instead it was suggested by Batygin & Brown (2016a) that the planet would cause similarities in both the argument of perihelia, ω, and the longitude of ascending node, Ω, through secular effects (Batygin & Morbidelli, 2017), which could then also account for other features seen in the Kuiper Belt (Batygin & Brown, 2016b).

Having another major planet in the outer Solar System would result in other observable effects. Both Gomes et al. (2016) and Bailey et al. (2016) suggested that the six-degree Solar obliquity could be explained as a natural result of the additional planet. And, as discussed by Fienga et al. (2016), a Planet 9 with a true anomaly of

$v \approx 120^\circ$ would significantly reduce the observed Cassini residuals. There have also been further studies which have reexamined the likeliness of an additional planet, with Cáceres & Gomes (2018) having suggested that in fact smaller perihelion distances provide better confinements.

However, there is still no consensus on whether the resonant perturbation mechanism is sufficient to describe the observed clustering of TNOs, what the most likely parameters of the planet are, or if it is in fact likely for there to be a planet at all (Batygin et al., 2019). There have been several alternative proposals for how such a clustering of TNOs could be explained, ranging from regular secular dynamics being sufficient (Beust, 2016), to the possibility of a primordial Black Hole (Scholtz & Unwin, 2020) which could have been captured instead of a free-floating planet. Furthermore, there are other difficulties of explaining such a Planet 9 as it's thought to be unlikely to have migrated into its current orbit, or to have been a captured free-floating planet (Parker et al., 2017).

Finally, it is also uncertain if the grouping of TNOs that the entire Planet 9 hypothesis was based on is actually due to observational bias (Bernardinelli et al., 2020a). Using the Outer Solar System Origins Survey (OSSOS) (Bannister et al., 2016), Shankman et al. (2017) discovered eight ETNOs which they claim have orbital parameter distributions consistent with what they would expect to detect and not grouped by a ninth Planet. Whereas Brown (2017) argues that the observed ETNOs must be grouped by external perturbations (Brown & Batygin, 2019). Sheppard et al. (2019) also suggest that an additional planet is still favoured, but concede that more studies would need to be done which fully take into account the selection functions of the various surveys used to observe the ETNOs. It's essential to enhance the current search and discover more ETNOs to place further constraints on the Planet 9 hypothesis. However, regardless of the existence of Planet 9, more TNOs need to be discovered to better understand the structure of the outer Solar System.

The Dark Energy Survey (DES), while constructed as a cosmological survey, is perfectly situated to discover faint objects in the outer Solar System with its repeat observations in a 5,000 square degree footprint, and its ability to identify very dim objects with a 10σ limiting magnitude of 23.2 in the r-band (Neilsen et al., 2016) using its powerful camera, DECam (Flaugher et al., 2015). Because a Planet 9 with a mass in the range $5 < M < 10M_e$ would have an aphelion magnitude of between $21.2 < V_{mag} < 24$ (Batygin et al., 2019), it should be detectable within the DES footprint. Indeed, DES and DECam have already been used to discover many TNOs, including two of the ETNOs first used to hypothesise the presence of Planet 9 (Trujillo & Sheppard, 2014, DES Collaboration, 2016, Gerdes et al., 2016, 2017, Becker et al., 2018, Khain et al., 2018, 2020, Lin et al., 2019, Bernardinelli et al., 2020b).

Our current process of detecting ETNOs and other distant objects in DES is to first combine observations of objects across different images. This is done by linking pairs of objects moving across images where the observed motion is consistent with Earth's parallax motion (Khain et al., 2020). These pairs can then be joined to give sets of three points in three different images taken on three separate nights. With these sets of three points, (which we call triplets), it's then possible to fit them to an orbit to determine if there can be a bounded orbit well defined by the 6 parameters: Semi-major axis, a; Eccentricity, e; Inclination, i, Longitude of ascending node Ω; Argument of perihelia, ω; and the Mean Anomaly, M (Bernstein & Khushalani, 2000).

Currently this process of orbit fitting is the slowest stage in our detection pipeline, and as the vast majority of triplets formed were from linking pixel-level fluctuations and artifacts that remained after difference imaging (Kessler et al., 2015) (which we refer to as noise), a lot of time is spent identifying this noise. In this paper we suggest an alternative method by implementing machine learning (ML), which is separate from the work done by Bernardinelli et al. (2020b), and Holman et al. (2018) who aimed to reduce the number of erroneous triplets that were initially linked. The machine learning classifier acts as an extra preprocessing stage to filter through the sets of triplets, identify and eliminate the majority of the triplets that result from noise in the data, and hence speed up the orbit fitting stage.

The first step of this process is to train the ML algorithms on simulations of ETNO triplets created using a survey simulator. Simulations are necessary as there simply aren't enough real observations of these distant objects to form a sufficient training set; however, it is possible to combine these simulations with real noise data to ensure the training data is representative.

Eight different supervised ML algorithms are trained and tested, each contained in the Scikit-Learn python package (Pedregosa et al., 2011), and we find that for this task of classifying rare events, the Random Forest classifier (RF) is the best performing algorithm. Once optimised and implemented in the detection pipeline, the RF allows for 80% of the noise triplets to be removed before performing orbit fitting which, as a result, runs five times faster.

In the following Sec 2.2 we describe the process of creating the simulated datasets that are used, and then summarise how to extract useful features. In Sec 2.3. we describe the ML algorithms tested and how the final classifier is optimised before giving the results in Sec 2.4. We then implement the classifier into the full search pipeline in Sec 2.5, and finally conclude our work in Sec 2.7.

2.2 Data Simulations and Feature Extraction

To be able to implement a supervised ML system, the ML algorithms first had to be trained and tested on data where the classifications were already known. As our problem focuses on finding rare events, there was not sufficient real data to form a large enough, and effective, training set to determine the algorithms' usefulness. Thus, synthetic objects were created and used to make simulated detections of ETNOs and possible Planet 9s within DES.

The simulations of detected ETNOs, and possible Planet 9s, were made using a survey simulator (Hamilton, 2019) which took an array of orbital parameters of objects given in Table 2.1, and by using these fake orbits, calculated whether or not each object could ever be visible in DES. This was done by calculating the limiting magnitude of each exposure within 7° of the position of the fake object at the beginning of DES operations (as even the fastest moving TNO couldn't have moved that far since the start of DES operations), and then project the position of each object into these nearby exposures to determine whether the object fell on a CCD during that exposure (Hamilton et al., 2021).

For the objects which could be detected, the simulator gave their positions in each DES image which could then be linked across multiple images. As the simulated objects were so distant, their motion across images was dominated by Earth's parallax motion, so pairs of objects could be found by linking the objects with motion consistent with the parallax motion. Pairs with common points were then combined to form triplets, sets of three points linked across three different images, as three points was the minimum requirement to perform an orbit fit to determine whether the observed points corresponded to an object or arose from noise in the images (see Sec 2.5 for a more complete description of the detection pipeline).

The majority of the dataset used was made up of real data which contained around 250000 triplets that had previously been linked but shown to result from noise after using the original method of performing a full orbit fit on every triplet detected. Although these real triplets could contain some small number of detections of objects which were misclassified, the vast majority were confidently due to noise, and the machine learning algorithms used shouldn't have been noticeably impacted. The real data acted as the sea of negatives, in which we searched for the much rarer positive triplets, of which around 10000 were made from the simulated objects. However, even with far more noise triplets compared to the triplets from simulated objects, this imbalance in the dataset was still less than would be observed in real data where over 99.9% of triplets result from noise.

With the data prepared the next important step was feature extraction, whereby the features which were used by the ML algorithms were selected. In the case of having many raw parameters, one of the main aims of feature extraction is to lower the dimensionality of the data. There are several ways to reduce the dimensionality but perhaps the common way this can be done is by using the coefficients of principal component analysis (PCA) (Pearson, 1901) as features instead of features taken

Table 2.1: Range of the 4 orbital parameters which were required by the Survey Simulator to create fake ETNOs and Planet 9s. In addition to these parameters the three further orbital parameters required to fully describe an orbit - Mean anomaly (M), argument of perihelion (ω), and longitude of ascending node (Ω), were also taken to have a uniform distribution. With these parameters the Simulator generated fake observations which could then be linked to generate the fake triplets used for the training data.

Parameter	Range
Semi-major axis, a	150 AU $< a <$ 1000 AU
Eccentricity, e	$0.1 < e < 1$
Inclination, i	$0^\circ < i < 90^\circ$
Absolute Magnitude, H	$1 < H < 10$

directly from the raw data. However, our specific problem had a very low dimensionality to begin with, and as such the process of feature extraction became more of a task to see what transformations could be made to the data to give the features which resulted in the best classifications.

The raw data output by DES and the simulator contained the positions on the sky of each possible object in the image along with the time of observation. The most basic features which could be used were therefore the positions of the object in each image and the times of observation, giving a total of nine features. However, the link-map which was used to link the observations and generate triplets that could arise due to TNOs only output the change in velocities of the object between the different observations (as this is what was used to determine if the object's motion was consistent with Earth's parallax motion). This meant that instead of using the ecliptic coordinates of the object in each image, the change in longitudinal and latitudinal velocities were used as features where the velocities were combined as

$$dvlon = \frac{vlon_{12} - vlon_{23}}{vlon_{12} + vlon_{23}}, \tag{2.1}$$

$$dvlat = \frac{vlat_{12} - vlat_{23}}{vlat_{12} + vlat_{23}}. \tag{2.2}$$

This reduced the initial nine features from the coordinates down to just two, but to include all the information about the trajectory of the object, the cosines of the

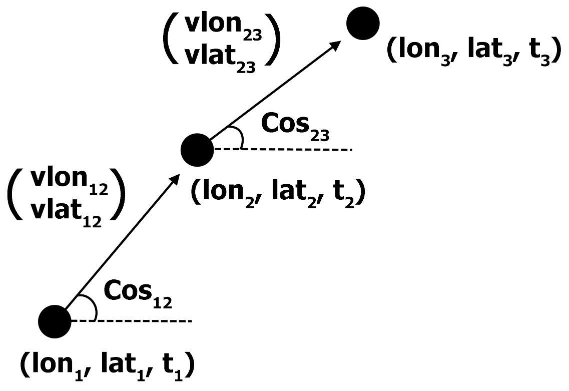

Figure 2.1: A triplet (displayed here to lie on a flat plane) was made by combining three points which had been linked across three different images taken on different nights. By transforming into ecliptic coordinates we were left with three sets of longitude and latitude as well as the times of the observations, resulting in 9 features. We then reduced the number of features by calculating the longitudinal and latitudinal velocities between each point, and further reduced these to simply the change in each velocity. By also using the two cosines between the two pairs of observations as features, we included all information needed about the trajectory of the object to be used by the machine learning algorithms to infer if the object could have a real orbit.

angles between points in the triplet were also included as features, giving the final four features used by the ML algorithms: the change in longitudinal and latitudinal velocities (dvlon, and dvlat), and the cosine of the angles between points ($\cos_{12}$, and $\cos_{23}$), which are displayed in Figure 2.1 of a triplet.

While these four features of the data were found to be sufficient to train ML algorithms to provide a significant increase in efficiency, we also investigated whether an even greater performance boost could be obtained using a dataset containing additional features. In section 2.6 we present the results from testing the exact same eight ML algorithms using a different dataset containing 80000 triplets from simulated objects. These simulations were made to also preserve the four longitudinal and latitudinal velocities to use as features rather than only using the two changes in velocities as in the original dataset.

2.3 Machine Learning Methodology

Having extracted the useful features, the next stage was to test various ML algorithms to determine which algorithm would give the best classification results. Here we perform tests on the following eight supervised ML algorithms with the aim of implementing a new ML stage to increase the efficiency of our TNO detection pipeline. The algorithms tested were: Logisitic Regression (LR), Naive Bayes (NB), Multi-layer Perceptrons (MLP), *k*-Nearest Neighbours (*k*NN), Decision Trees (DT), Boosted Decision Trees (BDT), Random Forests (RF), and Extremely Randomised Trees (ERT). Each of the classifiers, described in detail in section 1.4, were then tested using the method described below.

2.3.1 Methodology

To test each of these supervised ML algorithms, the data first had to be split into training, testing, and validation sets. We also performed cross-validation (Kohavi, 1995) across the train/test set to select the best combination of hyperparameters, the parameters which are used to build the ML algorithms (see Sec 2.3.2), and to be able to identify and minimise the impact of any possible overfitting. We used a random split taking 70% of the data to be used to train the algorithms, giving a healthy training set size with over 200000 triplets, and leaving the remaining 30% for tests and validation. However, due to the nature of the problem of looking for rare events, the datasets were incredibly imbalanced which could cause problems when trying to perform the classifications. Rather than forcibly making balanced datasets, which then wouldn't be representative of the true data, certain algorithms (LR, DT, RF, ERT) could take into account the class imbalance by applying a weighting to the data during the training.

Another common step taken before training any algorithm is to scale the data, making all the features have a range $0 < |\text{feature}| < 1$. This can be useful if the ML algorithm uses gradient descent, as convergence will occur much faster on normalised data (Johnson & Zhang, 2013). It can also be necessary if the data features have different units and varying ranges of values, as some models are sensitive to magnitude or use the euclidean distance between points such as *k*NN. However, scaling could also remove useful information if the difference in ranges of the features is important or resulting from some physical effect. Furthermore, scaling may not even be possible if the full range of data isn't known. In these tests, although scaling was also applied, it was found to make very little difference to the performance of the majority of the classifiers and was ignored when implementing the final RF classifier.

2.3.2 Hyperparameter Optimisation

The final step of creating the ML algorithms to be tested was to state the hyperparameters. Hyperparameters are the parameters of ML algorithms that get set prior to the learning process, and are used to create the algorithms, allowing all other parameters of the model to be learned from the training data. As an example, in random forests these hyperparameters include the number of estimators, which is the number of trees that make up the forest.

We tested the three different methods of optimisation: brute force optimisation, random optimisation, and Bayesian optimisation, (which are described in section 1.4.3) each of which required a grid of hyperparameters to be defined. This grid defines the hyperparameters and the range of values to be tested. The final thing needed to perform optimisation is the metric that is being optimised for. As described in the following section, the recall was the most informative metric for this problem of searching for rare objects and therefore we optimised the algorithms to give the best possible recall scores.

For the initial tests of all of the ML algorithms a simple brute force search was completed on the grids defined in Table 2.2. These values of hyperparameters were chosen to provide a wide enough range and ensure sufficient variation for each algorithm by changing the values of the hyperparameters which had the greatest effect. Although some hyperparameters could have continued to increased past the chosen upper limits, such as the number of estimators used in the ensembles of decision trees, we used a maximum which would provide a good estimate of performance without taking days to compute. Similarly, rather than testing every hyperparameter, to save time we only selected the ones with the greatest impact on the algorithm and the remaining hyperparameters not listed in Table 2.2 were kept at their default Scikit-Learn values. For a complete analysis benchmarking would be required to fully understand the trade off between the training and inference times and the other metrics obtained, however, this was beyond the scope of this work.

After the results of these tests were obtained (which are shown in Table 2.5 in Sec 2.4), and the Random Forest was selected as the best performing classifier, a more complete optimisation was performed. All three optimisation methods were tested on the larger grid given in Table 2.3 and the results are given in Table 2.4. While all three optimisation techniques were successful in improving the performance, the Brute force method did give the largest improvements. However, the differences were minimal, and the factor of 10 difference in the time taken compared to the other methods makes using one of the alternative methods more appealing for future implementations. Furthermore, while changing the hyperparameters does fine-tune the algorithm and improve classification results, the effect is far less than changing the data itself and to improve the results any further one would need to add features in the data processing stages.

2.3.3 Metrics

Once the classifiers had been trained and tested their performance had to be determined. There are various metrics that can be used for analysing ML algorithms, as described in section 1.4.2, the simplest of which is the classification accuracy. Although using the accuracy gave a quick way to determine how well a classifier performed, it wasn't particularly useful in the information it provided. As accuracy is simply the number of true predictions / total predictions, and in this case the majority of the data should be easily identified as a true negative, the null accuracy (predicting everything to be a negative result) was very high at 95%. This means that quoting an accuracy which sounds incredibly good can in fact still be a very poor classifier, as seen in some of our tests. Instead of using the accuracy, far more useful metrics can be obtained from the confusion matrix, a matrix of the true values against the predicted values (Manning et al., 2008).

The confusion matrix, such as the binary example in Table 1.3, allows for the

Table 2.2: Grids of hyperparameters that were searched when constructing each classifier for the initial tests to be able to compare each of the machine learning algorithms. Hyperparameters not mentioned in the table were kept at the default Scikit-Learn values. The hyperparameters that were selected to be used for each classifier are shown in bold, with the exception of NB in which the only hyperparameter which can be set are the prior probabilities which were left to be automatically adjusted according to the data.

Classifier	Hyperparameter	Array of Values
LR	"dual"	[False, **True**]
	"tol"	[**1e-7**, 1e-6, 1e-5, 1e-4]
	"C"	[1.0, 2.0, 3.0, **4.0**, 5.0]
*k*NN	"no. neighbors"	[**1**, 5, 10, 50, 100]
	"weights"	[**"uniform"**, "distance"]
	"leaf size"	[**1**, 5, 10, 50]
DT	"min. samples split"	[**2**, 5, 10, 50]
	"criterion"	["gini", **"entropy"**]
	"splitter"	[**"best"**, "random"]
BDT	"loss"	["exponential", **"deviance"**]
	"no. estimators"	[50, 100, 150, **200**]
RF	"no. estimators"	[10, 50, 100, **200**]
	"max. features"	[**"auto"**, 0.1, 0.4]
	"min. samples leaf"	[**1**, 5, 10, 20]
ERT	"no. estimators"	[10, 50, 100, **200**]
	"max. features"	[**"auto"**, 0.1, 0.4]
	"min. samples leaf"	[**1**, 5, 10, 20]
MLP	"hidden layer sizes"	[**1**, 10, 50, 100]
	"tol"	(**1e-3**, 1e-4, 1e-5)

Table 2.3: Grid of hyperparameters used by the three different techniques in the optimisation process for the Random Forest. For the Random and Bayesian optimisations only the upper and lower values were used to obtain a random value between the two limits, whereas for Brute force optimisation the specific values within the range also had to be stated. Additional hyperparameters not listed in the table were kept at the default Scikit-Learn values. The final hyperparameter values which gave the highest recall score are given in bold.

Hyperparameter	Array of Values
"no. estimators"	[1, 10, 50, 100, **200**]
"criterion"	['gini', **'entropy'**]
"max. features"	[**0.1**, 0.4, 0.9]
"min. samples split"	[**2**, 5, 10, 20]
"min. samples leaf"	[**1**, 5, 10, 20]
"min. weight fraction leaf"	[**0**, 0.4]
"bootstrap"	[True, **False**]

Table 2.4: Results from using the random forest classifier when optimised using the three different methods as compared to the default classifier given by Scikit-Learn.

Optimisation technique	None	Brute Force	Random	Bayesian
Time Taken (s)	0.00	43437.40	4654.01	7272.07
Accuracy	0.9891 ±0.0004	0.9912 ±0.0004	0.9907 ±0.0003	0.9903 ±0.0004
Recall	0.8588 ±0.0061	0.9000 ±0.0062	0.8976 ±0.0057	0.8965 ±0.0060
Precision	0.9129 ±0.0096	0.9265 ±0.0063	0.9139 ±0.0083	0.9085 ±0.0085
F1 score	0.8847 ±0.0023	0.9122 ±0.0042	0.9055 ±0.0014	0.9034 ±0.0037
AUC	0.9877 ±0.0026	0.9963 ±0.0011	0.9947 ±0.0009	0.9930 ±0.0011

other important metrics to be calculated from the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). The most useful metric in this case was the recall (or completeness / sensitivity). The recall gives the best measure of how many possible observations would be missed. For this problem we didn't want to have any FN which could have actually occurred due to a real object, and focused on optimising for the recall. However, improving the recall score came at the cost of decreasing the precision (or purity). And although we allowed for more FP, the precision had to also be kept as high as possible to not have too many FP which would make the machine learning method inefficient.

A combination of these two metrics, the F1 score, was used to show the balance between the recall and precision. The F1 score is the harmonic average of the recall and precision, and as such also has its best value at 1 and worst at 0, allowing us to quickly determine the classifiers' performances. Another very useful metric is the area under the curve (AUC) of the receiver operating characteristic curve (ROC curve) (Fawcett, 2006). The ROC curve was plotted with the True positive rate (TPR) against the false positive rate (FPR), resulting in a curve where the ideal result with AUC = 1 would be a straight line up and across. A ROC curve with each of the tested algorithms is shown in Figure 2.2.

2.4 Classification Results

The full results of the tests are given in Table 2.5 and all of the quoted results were obtained using 5-fold cross-validation to obtain a value which was unaffected by overfitting, and allowed for the standard deviation to be calculated. The relative usefulness is displayed in Figure 2.2, a single plot overlaying the ROC curves for each of the classifiers, as well as using box and whisker plots in Figure 2.3.

From these it's clear that some of the ML algorithms have completely failed to identify the triplets resulting from the fake objects. MLP found no TPs, and achieved the same as the null result of classifying everything as negative (triplets resulting from noise). LR performed similarly, classifying almost all triplets as negatives, and by falsely classifying some noise as positives (triplets resulting from

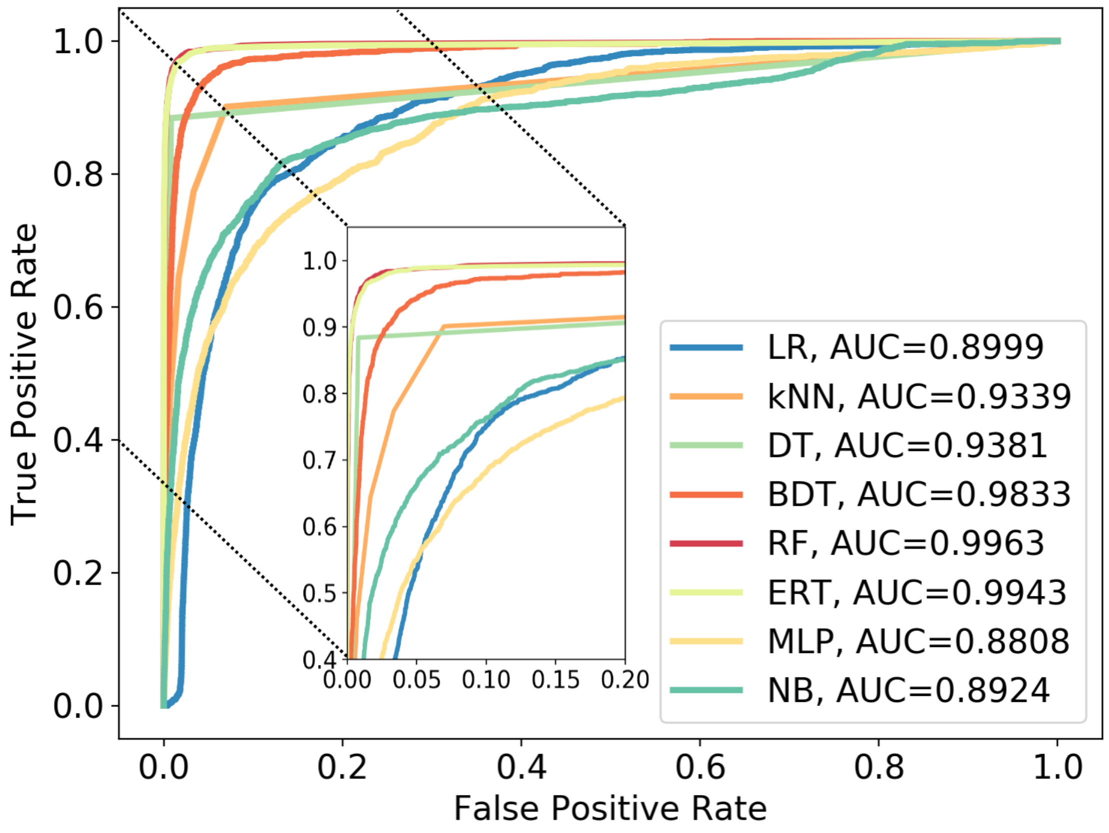

Figure 2.2: ROC curves for each machine learning classifier tested overlaid to be able to compare their effectiveness. The algorithms compared are: Logistic Regression (LR), *k*-Nearest Neighbours (*k*NN), Decision Trees (DT), Boosted Decision Trees (BDT), Random Forests (RF), Extremely-Randomised Trees (ERT), Multi-layer Perceptron (MLP), and Naive Bayes (NB). The tree-based classifiers outperformed all others, with the Random Forest (RF) and Extremely-Randomised Trees (ERT) being the best. The decision tree classifier (DT) produced a three-point curve as it outputs only the label rather than a predicted probability, giving only a single point of interest to plot.

Table 2.5: Results from testing the eight different machine learning algorithms described in section 1.4, the metrics were obtained using 5-fold cross-validation which also allowed for the standard deviation to be calculated and is given as the error.

Classifier	Logistic Regression (LR)	*k*-Nearest Neighbours (*k*NN)	Naive Bayes (NB)	Decision Tree (DT)	Boosted Decision Tree (BDT)	Random Forest (RF)	Extremely Randomised Trees (ERT)	Multi-layer Perceptron (MLP)
Accuracy	0.946 ±0.002	0.968 ±0.001	0.503 ±0.066	0.985 ±0.001	0.976 ±0.001	0.990 ±0.001	0.990 ±0.001	0.949 ±0.001
Recall	0.001 ±0.001	0.738 ±0.009	0.926 ±0.007	0.866 ±0.008	0.694 ±0.007	0.892 ±0.006	0.880 ±0.005	0.000 ±0.000
Precision	0.018 ±0.009	0.664 ±0.005	0.088 ±0.011	0.838 ±0.005	0.811 ±0.012	0.914 ±0.007	0.924 ±0.006	0.000 ±0.000
F1 score	0.002 ±0.001	0.690 ±0.004	0.160 ±0.019	0.852 ±0.005	0.747 ±0.007	0.902 ±0.002	0.902 ±0.004	0.000 ±0.000
AUC	0.900 ±0.006	0.934 ±0.004	0.892 ±0.007	0.938 ±0.003	0.983 ±0.001	0.996 ±0.001	0.994 ±0.001	0.881 ±0.006

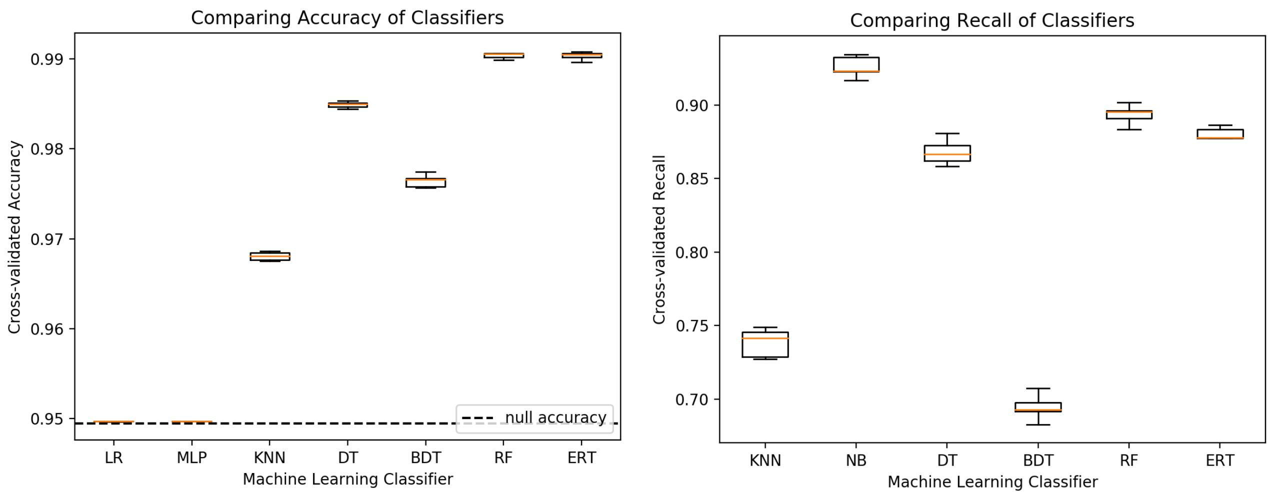

Figure 2.3: Box and Whisker plots comparing the accuracy and recall of the various different machine learning classifiers tested. The NB classifier is excluded from the accuracy graph as its value was so low (≈ 0.5), and similarly both LR and MLP are removed from the recall plot as their recalls were essentially 0 having recovered the null result of predicting all triplets to result from noise. The interquartile range (making up the box) was obtained by performing cross-validation in the tests of every algorithm, which also provided a standard deviation to be used as the uncertainty. The range of the results was shown by the extended ‘whiskers’, and the median is shown in red.

simulated objects), it had an accuracy lower than that of the null accuracy. While it's possible that spending more time optimising these algorithms could have resulted in improving them to no longer give the null classification, they also would have continued to classify too many FPs and had a precision too low to improve the efficiency of the search pipeline. Furthermore, compared with the other algorithms which were able to provide much better results with only the quick optimisation which was carried out on all the algorithms, they were not worth considering for this task.

The remaining classifiers all did much better, not having recall and precision scores close to 0, however, the tree-based classifiers seemed to be the best performing algorithms. Although *k*NN was somewhat successful, it had both a lower accuracy and precision/recall than most of the tree based methods, and didn't seem to be an optimal classifier for this problem. NB performed better than all other classifiers for the recall, but while this was the most important metric, it was only able to achieve this high a value by classifying almost half the data as positives, and as such it had a very low precision and by far the worst accuracy. The accuracy and precision being so low meant that it wasn't a useful classifier on its own, as it wouldn't be at all efficient when searching for objects; however, it could have been used if combined with other algorithms in a voting system, but this possibility is left for future work where we would consider more complex algorithms.

The tree-based classifiers were strong performers, but with some crucial differences between them. The basic DT classifier, although did well classifying the training set, was slightly overfitting despite the cross-validation and had lower metric scores which meant that it wouldn't be useful when applied to new data. The ensembles methods were much better at addressing this overfitting, but the BDT was consistently worse than the randomised methods due it not being able to handle the huge imbalance between classes. As a result, in all metrics, the boosted trees did worse than both the DT and forest classifiers, making it more similar to *k*NN in performance and also not useful for this problem. There was less to distinguish between the RF and ERT classifiers which had very similar metrics and performed very well at classifying the rare events; however, the RF was the faster method taking almost half the time to train and complete the classifications. On top of this the RF had a higher recall, suggesting that the additional stage of randomness in ERT was unnecessary for this problem.

Having selected the Random Forest as the most successful classifier, we then produced pair plots shown in figure 2.4 to examine the distributions of the features and suggest how the algorithm was able to produce its classifications. The majority of the simulated objects had quite sharp peaks due the fact that TNOs were more likely to have very small changes in longitudinal and latitudinal velocities and have cosines close to 1. Although one could have therefore used simple cuts to select the object closest to the peaks, by doing so far more triplets resulting from simulated objects would have been misclassified resulting in more possible detections getting missed. Instead, by implementing a machine learning algorithm such as the Random Forest it was possible to achieve far better classifications, and the tree-based algorithms might have performed better than others due to their nature of using many decision rules to be able to 'pick out' the majority of the simulated objects without also misclassifying much of the noise.

The final step taken to improve the performance of the Random Forest classifier was to change the decision threshold. In making classifications the RF calculated a predicted probability for each triplet to determine the probability of it resulting from noise or a real object. The decision threshold is the value at which the threshold is set so that for probabilities above this threshold the classification is taken to be positive (and the triplet results from a real object), and for probabilities below this threshold the classification is negative (and the triplet results from noise).

The default threshold was set $= 0.5$, however, as can be seen in Figure 2.5, which shows how the recall and precision change with the decision threshold, we were able to obtain a better result for the recall by lowering this threshold. Although lowering the threshold to our chosen value ≈ 0.2 resulted in a lower accuracy and precision, the recall improved sufficiently so that we were far less likely to miss a possible detection of a real object. Before changing the decision threshold the RF

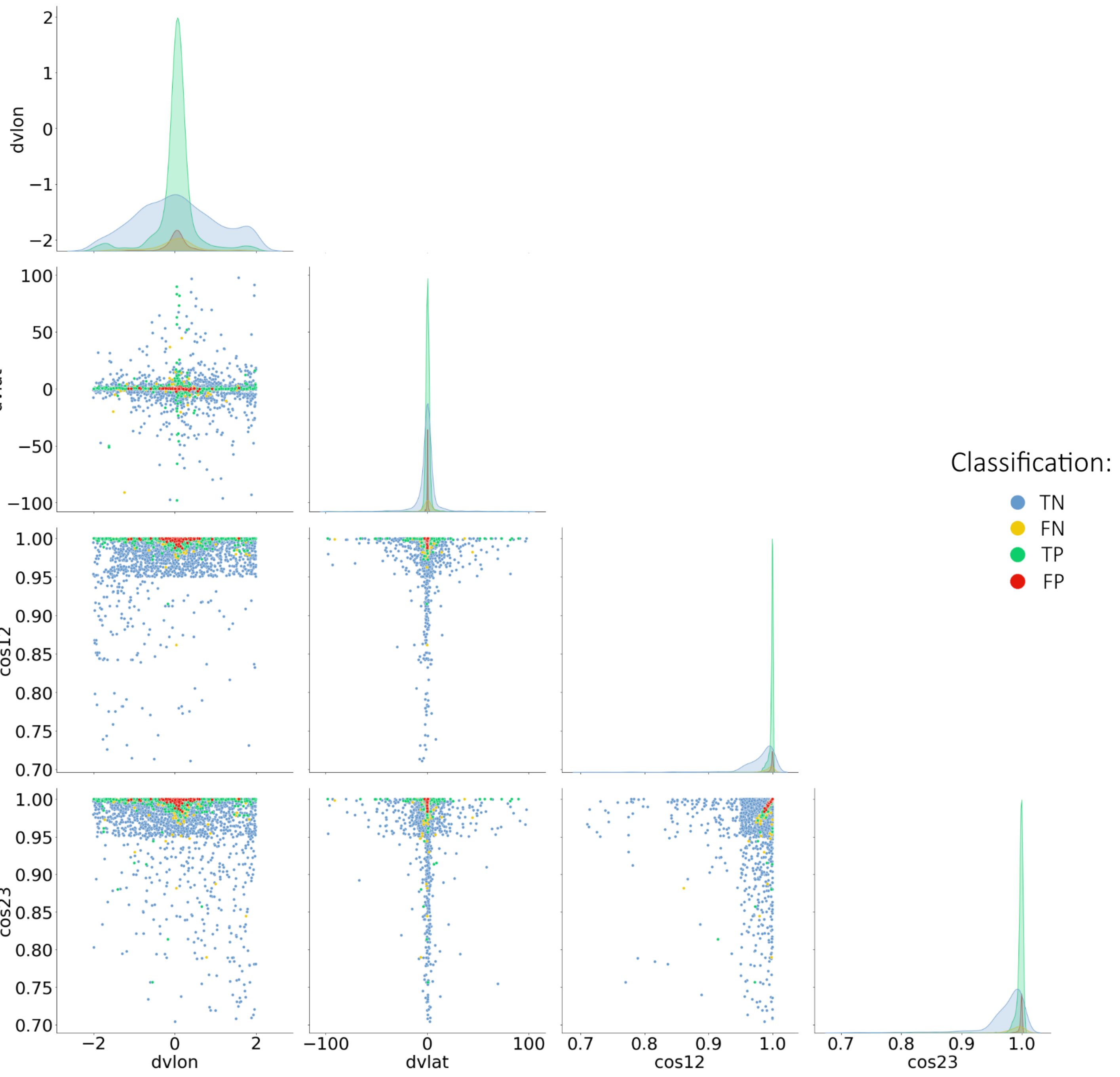

Figure 2.4: Pair plots showing the 4 features: dvlon, dvlat, $\cos_{12}$, and $\cos_{23}$, which were used by the Random Forest classifier plotted against each other with the label of their classification shown by the colour - blue for true negatives, yellow for false negatives, green for true positives, and red for false positives. Machine learning was especially beneficial given the overlap of the classes in each feature space meaning that there were no simple cuts able to separate the classes.

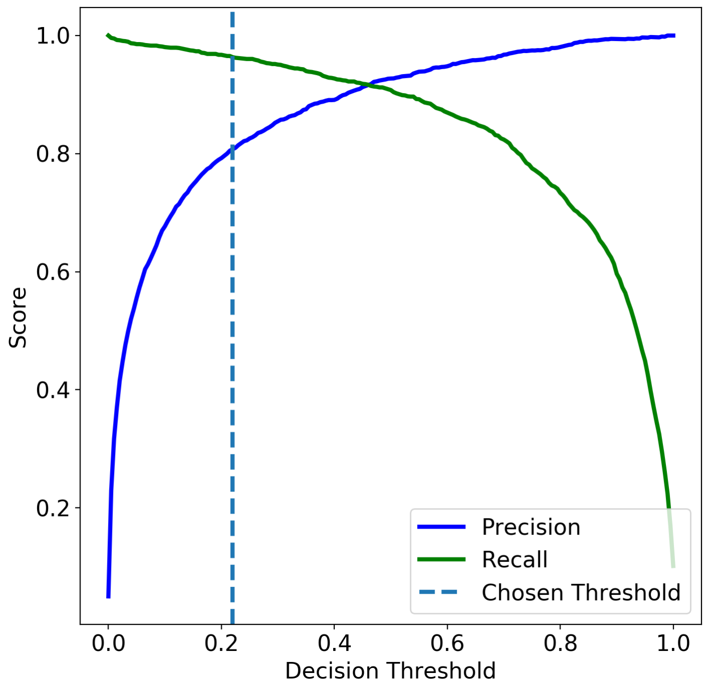

Figure 2.5: Graph showing the precision and recall scores of the Random Forest classifier as a function of the decision threshold. The default value of the threshold was set $= 0.5$, however, by changing to the chosen value ≈ 0.2 we were able to achieve a better recall without reducing the precision (and hence efficiency) by too much.

was missing 163 out of the 4600 (3.5%) triplets from simulated objects that were in the test set. Having changed the threshold, this was lowered to only 73/4600 (1.5%) of the triplets resulting from simulated objects being misclassified as noise, and although this does mean missing these triplets, in the full pipeline multiple triplets from the same object are almost always required to actually result in a confirmed detection. As such, although some triplets were missed, enough triplets were correctly classified that the vast majority of real objects would still be recovered.

2.5 Detection Pipeline

After the Random Forest had been found to be the best classifier, optimised, and had the decision threshold changed to further improve its performance, it was possible to implement the RF into the full search pipeline.

Our pipeline to identify TNOs can be described in three main stages which we summarise here, and a complete description of the entire process was done by

Bernardinelli et al. (2020b). First, the observational data had to be linked to give the sets of observations that could be of the same object. For each point in the data a linkmap was used to produce an array of all possible points that could be linked to it, determined by whether or not the motion between points seemed to be consistent with Earth's parallax motion. For TNOs the motion needs to be consistent with Earth's parallax as they are such distant objects that their proper motion is much less apparent than the motion of the Earth. The output of the linkmap results in pairs of points that could possibly be the same object, and the next step was to take the linked pairs and form triplets, the sets of three points that could all be from the same object. This was done in the same way that the pairs were formed, checking to see if the motion from one set of pairs to the next was consistent.

Once the triplet was formed it needed to be checked to see if it could have actually arisen from an object or if it was an artifact of noise in the data (Kessler et al., 2015). This was where the ML classifier was implemented as an extra preprocessing step to quickly discard the majority of the triplets which result from noise in the data. After the majority of the noisy triplets had been removed (over 80%), the remaining triplets were fitted to an orbit to see if they could be described by a real orbit of an object, or if they were still due to noise. This orbit fitting stage determined whether there could be a bounded orbit well described by the six orbital parameters: Semi-major axis, a; Eccentricity, e; Inclination, i, Longitude of ascending node Ω; Argument of perihelia, ω; and the Mean Anomaly, M (Bernstein & Khushalani, 2000), and was the slowest stage of the pipeline. By removing most of the noise using the ML classifier rather than orbit fitting every triplet generated during linking, most of this computationally expensive step was saved, and the search was sped up significantly.

The increase in efficiency by implementing ML was evident and the pipeline was run five times faster when using the classifier. This was achieved as, out of the dataset containing around a quarter of a million triplets, only around 10% were kept to be fitted to an orbit, but even after keeping so few of the initial triplets, the classifier only misclassified 73/4600 (1.5%) triplets from simulated objects. Although this is still missing 1.5% of the triplets arising from objects, real objects will be almost always be discovered by multiple triplets. As such having a recall ≈ 0.96 which was achieved by the Random Forest is likely to recover the vast majority of the real objects. This would allow for the edited pipeline which includes the classifier to be run as a quick preliminary search and still be able to detect most of the objects before completing a full analysis.

2.6 Additional Features

An additional dataset was created to test the effects on the various ML algorithms of using different features. The new data created contained the longitudinal and latitudinal velocities (vlon_{12}, vlon_{23}, vlat_{12}, vlat_{23}) as well as the cosines of the angles between the pairs of points in the triplet ($\cos_{12}$, and $\cos_{23}$) which had previously been used as features. Using these six features instead of the original four, the same methodology was followed and the results are given in table 2.6, along with the results when using the original features (dvlon, dvlat, $\cos_{12}$, and $\cos_{23}$) on the new data for a fair comparison. To visualise the new feature space, pair plots were also produced for the best performing random forest which are shown in figure 2.6.

As can be seen from the results in table 2.6, in general the algorithms were able to achieve slightly improved metrics from the addition of the extra features. This improvement suggested that it would be worthwhile to include further features to provide as much information as possible to the machine learning methods. However, as can be seen from figure 2.6, the pair plots showed that there were still no distinct decision boundaries and the classification problem of extracting the objects from noise remained complex. Furthermore, to be able to include additional features such as the raw coordinates of the observations, the link-map used to generate the triplets would need to be changed to preserve this information.

2.7 Summary

The classification of rare events, such as this example of searching for ETNOs and a possible ninth Planet, has become an even more important venture in light of the vast datasets becoming available. In the wake of future surveys such as the Vera

Table 2.6: Results from testing the eight machine learning algorithms on a new dataset using the six features ($vlon_{12}$, $vlon_{23}$, $vlat_{12}$, $vlat_{23}$, cos_{12}, and cos_{23}). The results of using the original 4 features (dvlon, dvlat, cos_{12}, and cos_{23}) on the same data are also presented to compare the effects of using the different velocities as features.

	6 Features ($vlon_{12}$, $vlon_{23}$, $vlat_{12}$, $vlat_{23}$, cos_{12}, and cos_{23})					4 Features (dvlon, dvlat, cos_{12}, and cos_{23})				
	Accuracy	Recall	Precision	F1	AUC	Accuracy	Recall	Precision	F1	AUC
Logistic Regression (LR)	0.729 ± 0.217	0.408 ± 0.243	0.106 ± 0.023	0.157 ± 0.022	0.701 ± 0.197	0.690 ± 0.003	0.876 ± 0.002	0.138 ± 0.012	0.239 ± 0.009	0.853 ± 0.010
k-Nearest Neighbours (*k*NN)	0.988 ± 0.001	0.927 ± 0.001	0.863 ± 0.001	0.894 ± 0.001	0.991 ± 0.001	0.970 ± 0.001	0.744 ± 0.001	0.730 ± 0.001	0.737 ± 0.001	0.967 ± 0.001
Naive Bayes (NB)	0.716 ± 0.187	0.961 ± 0.021	0.159 ± 0.042	0.273 ± 0.051	0.888 ± 0.063	0.899 ± 0.006	0.833 ± 0.003	0.337 ± 0.014	0.480 ± 0.008	0.941 ± 0.005
Decision Tree (DT)	0.995 ± 0.001	0.970 ± 0.002	0.948 ± 0.003	0.959 ± 0.002	0.983 ± 0.001	0.986 ± 0.002	0.892 ± 0.002	0.865 ± 0.003	0.878 ± 0.002	0.941 ± 0.001
Boosted Decision Tree (BDT)	0.993 ± 0.002	0.909 ± 0.003	0.970 ± 0.007	0.939 ± 0.005	0.998 ± 0.002	0.979 ± 0.003	0.710 ± 0.006	0.953 ± 0.002	0.924 ± 0.004	0.988 ± 0.001
Random Forest (RF)	0.998 ± 0.001	0.975 ± 0.001	0.991 ± 0.001	0.983 ± 0.001	0.999 ± 0.001	0.992 ± 0.001	0.899 ± 0.001	0.953 ± 0.002	0.933 ± 0.002	0.998 ± 0.001
Extremely Randomised Trees (ERT)	0.998 ± 0.001	0.968 ± 0.002	0.990 ± 0.002	0.979 ± 0.002	0.999 ± 0.001	0.992 ± 0.001	0.893 ± 0.003	0.974 ± 0.003	0.932 ± 0.003	0.995 ± 0.001
Multi-layer Perceptron (MLP)	0.961 ± 0.003	0.574 ± 0.098	0.699 ± 0.064	0.620 ± 0.035	0.969 ± 0.002	0.957 ± 0.001	0.443 ± 0.067	0.679 ± 0.023	0.533 ± 0.043	0.962 ± 0.001

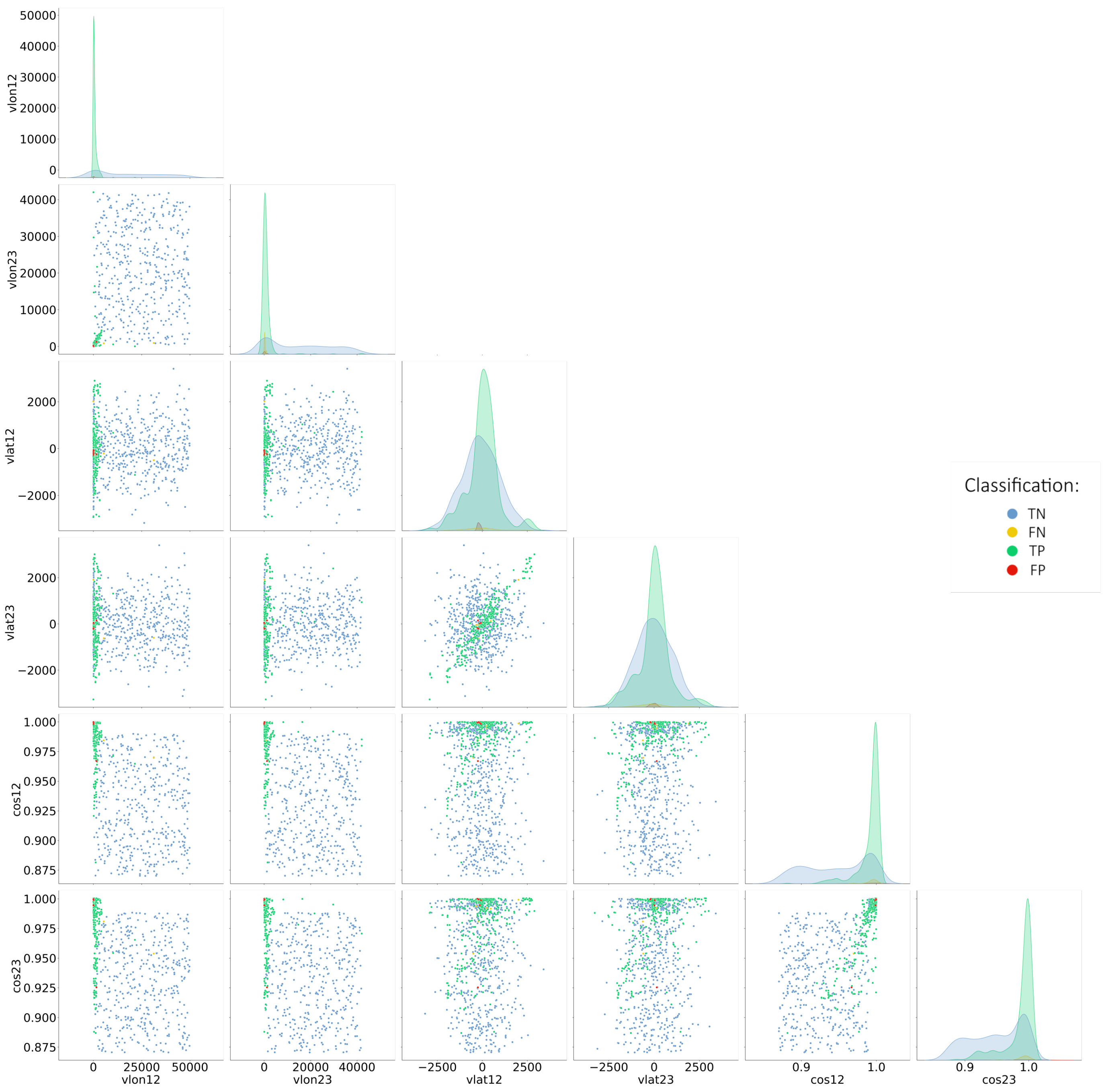

Figure 2.6: Pair plots showing the classifications made by the Random Forest on the new data using the six features: vlon_{12}, vlon_{23}, vlat_{12}, vlat_{23}, $\cos_{12}$, and $\cos_{23}$. The label of the classification was shown by the colour - blue for true negatives (noise correctly predicted as noise), yellow for false negatives (objects incorrectly predicted as noise), green for true positives (objects correctly predicted as objects), and red for false positives (noise incorrectly predicted as objects).

C. Rubin Observatory (or Large Synoptic Survey Telescope (LSST)), which will produce 10 million transient events every night, being able to utilise ML methods will be vital to improve efficiencies and allow further analysis to be undertaken.

In this work we've shown that implementing ML classifiers using the very user friendly package scikit-learn could be used as a preprocessing step, removing the vast majority of erroneous detections, which helped speed up our discovery pipeline. Having tested eight of the most used algorithms we discovered that the Random Forest classifier was the best performing overall, and had the best functionality of being less prone to overfitting and taking into account imbalanced datasets.

Our results showed that the optimised Random Forest used could perform incredibly well, and achieved an AUC = 0.996. Furthermore, by changing the decision boundary we maximised the recall, giving a recall $= 0.96$ to ensure that the vast majority of the triplets resulting from real objects could be recovered. We also maintained a high accuracy and precision at 0.99 and 0.80 respectively. This meant that our method was far more efficient, preventing the vast majority of the triplets resulting from noise from advancing to the orbit fitting stage, and greatly speeding up the pipeline.

If used in parallel with the existing pipeline which fits all triplets to an orbit to ensure it's 100% complete, implementing machine learning could allow for a useful preliminary search to identify objects more quickly and provide a cross check for the objects passing the orbit fitting.

The work presented here opens the door for analyses on searching for other populations of TNOs in DES data. This method of using machine learning to filter noise could be especially useful to help identify closer objects where the faster motion results in even more noise. It would be desirable to investigate whether the RF classifier would be as effective when applied to these different populations of objects, and implement a ML method at a similar stage in the detection pipeline.

Further investigation could also be done to implement new algorithms which have the potential to speed-up the pipeline even more, as well as using machine learning in other areas, such as changing the way that points can be linked through images and allowing additional features to be used, which will make it possible to further improve the current search. Improvements such as these will aid the discovery of far more of the TNO population which is crucial information for constraining Planet 9 and learning more about our Solar System.

Chapter 3

Benchmarking Machine Learning Methods for Photometric Redshift Estimation

This chapter is based on Henghes et al. (2021): Benchmarking and scalability of machine learning methods for photometric redshift estimation, Monthly Notices of the Royal Astronomical Society (MNRAS), Volume 505, Issue 4, August 2021, Pages 4847 - 4856.

Obtaining accurate photometric redshift (photo-z) estimations is an important aspect of cosmology, remaining a prerequisite of many analyses. In creating novel methods to produce photo-z estimations, there has been a shift towards using machine learning techniques. However, there has not been as much of a focus on how well different machine learning methods scale or perform with the ever-increasing amounts of data being produced. Here, we introduce a benchmark designed to analyse the performance and scalability of different supervised machine learning methods for photo-z estimation. Making use of the Sloan Digital Sky Survey (SDSS - DR12) dataset, we analysed a variety of the most used machine learning algorithms. By scaling the number of galaxies used to train and test the algorithms up to one million, we obtained several metrics demonstrating the algorithms' performance and scalability for this task. Furthermore, by introducing a new optimisation method, time-considered optimisation, we were able to demonstrate how a small concession of error can allow for a great improvement in efficiency. From the algorithms tested we found that the Random Forest performed best with a mean squared error, $MSE = 0.0042$; however, as other algorithms such as Boosted Decision Trees and k-Nearest Neighbours performed very similarly, we used our benchmarks to demonstrate how different algorithms could be superior in different scenarios. We believe benchmarks like this will become essential with upcoming surveys, such as the Vera C. Rubin Observatory's Legacy Survey of Space and Time (LSST), which will capture billions of galaxies requiring photometric redshifts.

3.1 Introduction

Calculating distances to cosmological objects remains one of the most important steps required for probing cosmology. These distances are given by the distance-redshift relation, and hence one needs very accurate measures of redshift to be confident in the inferred distances. Ideally, high resolution spectra would be obtained for every object enabling for a precise measurement of the redshift. However, with current and future surveys such as the Dark Energy Survey (DES) (DES Collaboration, 2005, 2016), Euclid (Amendola et al., 2018), and the Vera C. Rubin Observatory's Legacy Survey of Space and Time (LSST) (Tyson et al., 2003, Ivezić et al., 2019), even with large spectroscopic surveys such as the Dark Energy Spectroscopic Instrument (DESI) (Martini et al., 2018), only tens of millions of the galaxies will have spectroscopy performed, despite hundreds of millions of galaxies being observed.

In the absence of real spectroscopic measurements, obtaining photometric redshifts (photo-z) estimations is the only viable route available for scientists. There are two major techniques used for photometric redshift estimation, template flitting (e.g. Benitez (2000)) and machine learning (ML) (e.g. Collister & Lahav (2004)). Both methods rely on the photometric information produced by the survey, usually given as magnitudes in different colour bands. These magnitudes act as approximate measures of the underlying spectral energy distribution (SED) of the observed object, and by appropriately reconstructing the SED, a corresponding redshift can be inferred (Bolzonella et al., 2000).

Template fitting methods use a small and fixed set of template spectra for the estimations, and inherently relies on the assumption that the best fitting SED template provides the true representation of the observed SED. There are benefits of template methods, including the ability to incorporate physical information, such as dust extinction, into the model. However, embedding such physical constraints requires very precise calibration and an accurate model (Benitez, 2000).

Machine learning techniques, on the other hand, do not have any explicit model for capturing the physical information of the objects or of the estimation process.

Instead, ML techniques rely on a training dataset with spectroscopic redshifts from observed or simulated (or a combination of both) data for inferring an estimation model. More specifically, supervised learning models rely on a guided principle, that with sufficient examples of input-output pairs an estimation model can be inferred by understanding the latent variables of the process. In other words, ML methods derive a suitable functional mapping between the photometric observations and the corresponding redshifts.

The learning process relies on a labelled dataset consisting of a set of magnitudes in each wavelength band (the inputs) and corresponding true values of the spectroscopic redshifts (the output labels or ground-truth). The learning model, such as a random forest or neural network, learns the mappings which can be nonlinear. It has been shown that the functional mapping learned through the supervised learning can for some science goals outperform the template-based methods (Abdalla et al., 2011).

Although the usage of ML in handling this problem has become very common (Pasquet et al., 2019, D'Isanto & Polsterer, 2018, Hoyle, 2016), and some studies are beginning to investigate the efficiency of different models (Schmidt et al., 2020, Euclid Collaboration et al., 2020), there is still no comprehensive study outlining the benchmarking process and how it changes our overall understanding of how different ML methods handle the photo-z problem. In fact, this is a common problem across all domains of sciences, and as such, the notion of artificial intelligence (AI) benchmarking is an upcoming challenge for the AI and scientific community. This is particularly true in light of recent developments in the ML and AI domains, such as the deep learning revolution (Sejnowski, 2018), technological development on surveys (Dewdney et al., 2009), the ability to generate or simulate synthetic data (Springel, 2005), and finally the progress in computer architecture space, such as the emergence of GPUs (Kirk, 2007).

The notion of benchmarking (Dongarra et al., 2003) has conventionally been about how a given architecture (or an aspect of a given architecture) performs for a given problem, such as the LINPACK challenge (Dongarra et al., 1979). However, in our case, the focus is broader than just performance. Our motivation here is many-fold, including understanding how different ML models compare when estimating the redshifts, how these techniques perform when the available training data is scaled, and finally how these techniques scale for inference. Furthermore, one of the key challenges here is the identification of appropriate metrics or figures of merit for comparing these models across different cases.

We intend to answer some of these questions by introducing this as a representative AI benchmarking problem from the astronomical community. The benchmarks will include several baseline reference implementations covering different ML models and address the challenges outlined above. The rest of this chapter is organised as follows: In Section 3.2, we describe the the dataset used and include discussions on the features selected. In Section 3.3, we briefly describe the machine learning models that were evaluated in the study, followed by the descriptions of the optimisation and benchmarking processes and the different metrics that are part of our analyses. The results are then presented in Section 3.4 along with our observations, and we conclude the paper in Section 3.5 with directions for further work.

3.2 Data

The data used in our analysis comes entirely from the Sloan Digital Sky Survey (SDSS) (York et al., 2000). Using its dedicated 2.5 meter telescope at Apache Point Observatory (Gunn et al., 2006), SDSS is one of the largest public surveys with over 200 million photometric galaxies and 2 million useful galaxy spectra as of data release 12 (DR12) (Alam et al., 2015).

In this work we downloaded 1,639,348 of these galaxies with spectroscopic data available to be used by the machine learning algorithms. The spectroscopic redshift was required as it was taken to be the ground truth for the redshift that the algorithms were trying to predict using the magnitudes of each galaxy. SDSS took images using five different optical filters (u, g, r, i, z), and as a result of these different wavelength bands, there were five magnitudes for each observed galaxy

(Eisenstein et al., 2011).

The 1.6 million galaxies used in this investigation were from a cleaned dataset where it was a requirement for all five magnitudes to have been measured. In many cases for observations of galaxies there could be a missing value in one of the filters which would negatively impact its redshift estimation. By only using galaxies with complete photometry we ensured that our comparison of methods wasn't also being affected by the kind of galaxies within the different sized datasets.

Furthermore, the redshift range of the galaxies used was constrained to only have galaxies with a redshift, $z < 1$. While this greatly simplified the task of obtaining photo-z estimations and meant that the specific models tested wouldn't be useful outside of this range, there are far fewer galaxies with measured spectroscopic redshifts greater than 1, and we kept within this range to ensure that the training set would be representative of the SDSS spectroscopic sample. While this allowed for reliable estimates to be generated for SDSS galaxies inside this sample, it is not representative of the overall survey as there are far more galaxies without spectroscopy, and to allow for reliable estimates outside of this region, further steps would have to be taken to generalise the dataset. However, using this cleaned spectroscopic sample meant that the benchmarking performed could be carried out without also having to take into account the effects that an unclean dataset might have had on the different machine learning algorithms.

The main features of the data used by machine learning algorithms were the five magnitudes which could also be combined to give the four colours that are simply the difference in magnitudes between neighbouring wavelength bands (u-g, g-r, r-i, and i-z). There were additional feature columns contained in the SDSS data which could have been added such as the subclass of galaxy or the Petrosian radius (Petrosian, 1976, Soo et al., 2017). However; adding these additional features wouldn't have had a large impact on the results and could have added more issues due to incompleteness if the feature wasn't recorded for every galaxy. Instead it was decided to use only the information from the five magnitudes as features which we knew to be complete.

Finally, we also scaled the features by subtracting its mean and dividing by its standard deviation to give unit variance. This ensured that the machine learning algorithms used weren't being influenced by the absolute size of the values, where a difference in a feature's variance could result in it being seen as more important than other features. And by randomly splitting the full dataset to form the training and testing sets, the subsets created kept the same distribution of redshift and were representative of the overall dataframe.

3.3 Methodology

With the data prepared, the first step of the machine learning process was to split the entire dataset to create a training set, testing set, and validation set, whereby the test and validation sets were kept unseen by the machine learning algorithms until after it had been trained using the training data. As part of the benchmarking process the machine learning algorithms (listed in Sec 3.3.1) were trained and tested on many different sizes of datasets, and to do this the data was split randomly for each size of training and testing set required.

During training, the algorithms were also optimised by changing the hyper-parameters (which are the parameters of the models that control how the machine learning algorithms create their mappings from the features to the redshift). The most complete way of optimising would be to perform brute force optimisation where every combination of a defined grid of hyperparameters would be tested. However, this is far more computationally intensive than random optimisation which instead tests a random subset of the hyperparameter grid and provides a good estimate of the best hyperparameters. The grids of hyperparameters tested for each algorithm is given in Table 3.1 along with the selected parameters.

To be able to optimise the algorithms the decision first had to be made of which metric would be optimised for. There are three main metrics used for regression problems such as this: mean squared error (MSE), mean absolute error (MAE), and R squared score (R^2), where the formulae for calculating each of these metrics is given in section 1.4.2.

Table 3.1: Grids of hyperparameters that were searched to test and compare each machine learning algorithm, along with the hyperparameters that were selected by the random optimisation. The arrays of hyperparameters were chosen to give a good overview of different possible configurations of the algorithms, and by changing the parameters which had the greatest impact on the algorithms, we ensured finding a good representation of the 'best' performing algorithms.

Classifier	Hyperparameter	Array of Values Searched	Selected Value
LR	"fit intercept"	[True, False]	**True**
	"normalize"	[True, False]	**True**
*k*NN	"no. neighbors"	[1, 200]	**21**
	"weights"	["uniform", "distance"]	**"distance"**
	"leaf size"	[10, 100]	**27**
	"p"	[1, 4]	**2**
DT	'max. features"	[1, 5, "auto"]	**"auto"**
	"min. samples split"	[2, 100]	**38**
	"min. samples leaf"	[1, 100]	**64**
	"min. weight fraction leaf"	[0, 0.4]	**0**
	"criterion"	["mse", "mae"]	**mse**
BDT	"no. estimators"	[1, 200]	**88**
	"loss"	["ls", "lad", "huber", "quantile"]	**"lad"**
	"max. features"	[1, 5]	**4**
	"max. depth"	[1, 20]	**17**
	"min. samples split"	[2, 100]	**46**
	"min weight fraction leaf"	[0, 0.4]	**0**
RF	"no. estimators"	[1, 200]	**94**
	"max. features"	[1, 5]	**4**
	"min. samples leaf"	[1, 100]	**8**
	"min. samples split"	[2, 100]	**13**
	"min weight fraction leaf"	[0, 0.4]	**0**
	"criterion"	["mse", "mae"]	**mae**
ERT	"no. estimators"	[1, 200]	**147**
	"max. features"	[1, 5]	**4**
	"min. samples leaf"	[1, 100]	**3**
	"min. samples split"	[2, 100]	**87**
	"min weight fraction leaf"	[0, 0.4]	**0**
	"criterion"	["mse", "mae"]	**mse**
MLP	"hidden layer sizes"	[(100, 100, 100), (100, 100), 100]	**(100, 100, 100)**
	"activation"	["tanh", "relu"]	**"tanh"**
	"solver"	["sgd", "adam"]	**"adam"**
	"alpha"	[0.00001, 0.0001, 0.001, 0.01]	**0.01**
	"tol"	[0.00001, 0.0001, 0.001, 0.01]	**0.00001**
	"learning rate"	["constant","adaptive"]	**"constant"**

However; there are three additional metrics that are commonly used to determine the performance of photometric redshift estimations. These metrics are the bias, precision, and outlier fraction, which are all calculated from the residuals, Δz, defined as

$$\Delta z = \frac{z_{\text{pred}} - z_{\text{spec}}}{1 + z_{\text{spec}}}. \tag{3.1}$$

The bias is simply defined as the mean of these residuals, given by

$$\text{Bias} = < \Delta z > . \tag{3.2}$$

The precision (also 1.48× median absolute deviation (MAD) as defined by Ilbert et al. (2006)) gives the expected scatter and is defined as

$$\text{Precision} = 1.48 \times \text{median}(|\Delta z|). \tag{3.3}$$

Finally, the outlier fraction is the fraction of predictions where the residuals are greater than a set threshold (here chosen to be > 0.10), and is given by

$$\text{Outlier Fraction} = \frac{N(|\Delta z|) > 0.10}{N_{\text{total}}}. \tag{3.4}$$

Each of these metrics were also calculated and the results are given in Table 3.2.

As well as deciding which metric to optimise for, we introduced an extra stage included in the optimisation which allowed for a time-considered optimisation (see section 3.3.2). We optimised the machine learning algorithms for MSE (aiming to minimise the MSE) and used a random optimisation with 1000 iterations to ensure a good estimate of the best hyperparameters for each algorithm. Furthermore, we used a 3-fold cross validation (Breiman & Spector, 1992) to ensure that the algorithms weren't overfitting (which could mean that the algorithms were able to perform well for the training data used but then fail to generalise), and that the results would be valid for any given dataset. Once optimised each algorithm was then retrained and tested to give the final results given in Sec 3.4, along with the benchmarking results, where the benchmarking process used is described in 3.3.3.

3.3.1 Machine Learning Algorithms Tested

The algorithms selected were: Linear Regression, k-Nearest Neighbours, Decision Trees, Boosted Decision Trees, Random Forests, Extremely Randomised Trees, and a Multi-layer Perceptron. These algorithms are described in detail in section 1.4 and were chosen for testing as they are some of the most widely used machine learning algorithms which were all available through the python package Scikit-Learn (Pedregosa et al., 2011). While a simple neural network (Multi-layer Perceptron) was included, we didn't include any other examples of deep learning. This decision was made as deep learning algorithms perform best with many features (often thousands) and there's only so much information that the photometry could provide with the five magnitude features. Furthermore, it's been shown by Hoyle (2016) that 'traditional' algorithms can perform equally well as deep learning methods, and that it might only be beneficial to use more computationally expensive deep learning models when directly using images as the training data (Pasquet et al., 2019).

3.3.2 Time-Considered Optimisation

In the normal process of optimising machine learning algorithms, a single metric is chosen to minimise. If brute force optimisation is used, this produces an algorithm configured with the hyperparameters from the defined grid which gives the best result for the metric (e.g. the lowest MSE). Although this algorithm by definition would have the best result, it doesn't necessarily result in the most useful or suitable algorithm. The hyperparameters selected to minimise the error likely also act to increase the computational time required both in training and inference, resulting in a much slower model.

Rather than minimising a single metric, in time-considered optimisation we also consider the time taken by the models both in training and inference. By setting an error tolerance we allow for the model selection to suggest an alternative to the 'best' model (the model which minimises the error metric), instead providing a model which will have a higher error, while kept below the tolerance, but in return will also have faster training and inference times. This was done by optimising for a combination of MSE and time, taking the model which ran the fastest and that

Table 3.2: Results of testing the seven machine learning algorithms described in Sec 3.3.1. Each algorithm was trained using 10000 galaxies and tested using 5-fold cross validation to obtain the quoted standard deviation.

	Linear Regression (LR)	*k*-Nearest Neighbours (*k*NN)	Decision Tree (DT)	Boosted Decision Tree (BDT)	Random Forest (RF)	Extremely Randomised Trees (ERT)	Multi-layer Perceptron (MLP)
MSE	0.005714 ±0.000577	0.004438 ±0.000417	0.004631 ±0.000407	0.004277 ±0.000394	0.004221 ±0.000423	0.004327 ±0.000419	0.004701 ±0.000499
MAE	0.050931 ±0.001679	0.040881 ±0.001626	0.041827 ±0.001452	0.038757 ±0.001514	0.038504 ±0.001484	0.040459 ±0.001537	0.051260 ±0.008874
R^2	0.865198 ±0.009009	0.895208 ±0.007215	0.890677 ±0.006415	0.899017 ±0.006822	0.900366 ±0.007373	0.897861 ±0.007198	0.871507 ±0.014329
Bias	0.003132 ±0.000067	0.001498 ±0.000118	0.002543 ±0.000174	0.001731 ±0.000152	0.002310 ±0.000103	0.002343 ±0.000138	−0.00416 ±0.005911
Precision	0.043421 ±0.000578	0.031836 ±0.000845	0.032895 ±0.001137	0.028986 ±0.000264	0.029279 ±0.000766	0.031945 ±0.000609	0.040837 ±0.003310
Outlier Fraction	0.060500 ±0.005187	0.034800 ±0.007033	0.033400 ±0.007439	0.029300 ±0.005183	0.029700 ±0.004389	0.033400 ±0.006304	0.037600 ±0.002709

had an error below the set tolerance level. In certain cases, such as training the Decision Tree, it was possible to achieve a two magnitude increase in efficiency while increasing the error by $< 10\%$.

For the purpose of benchmarking the machine learning algorithms in this paper, we set the error tolerance to machine precision (which is usually 10^{-16}) resulting in the 'best' model in terms of error. This decision was made as these optimised algorithms would result in the algorithms most commonly used in other machine learning studies where time-considered optimisation hasn't be implemented.

3.3.3 Benchmarking

The benchmarking performed was achieved by recording the system state (described by the time, CPU usage, memory usage, and disk I/O) throughout the process of running the machine learning algorithms. This allowed us to compare the efficiency of both training and predicting performance of the machine learning models, and when combined with the regression errors obtained, allowed for a complete description of the performance of the different methods.

Our main focus of the benchmark was to investigate how training and testing times varied with different sizes of dataframes, and how the final redshift estimations would be affected. As such we incrementally changed both the training and testing datasets and recorded the times taken which allowed us to produce the plots shown in figures 3.1 - 3.5.

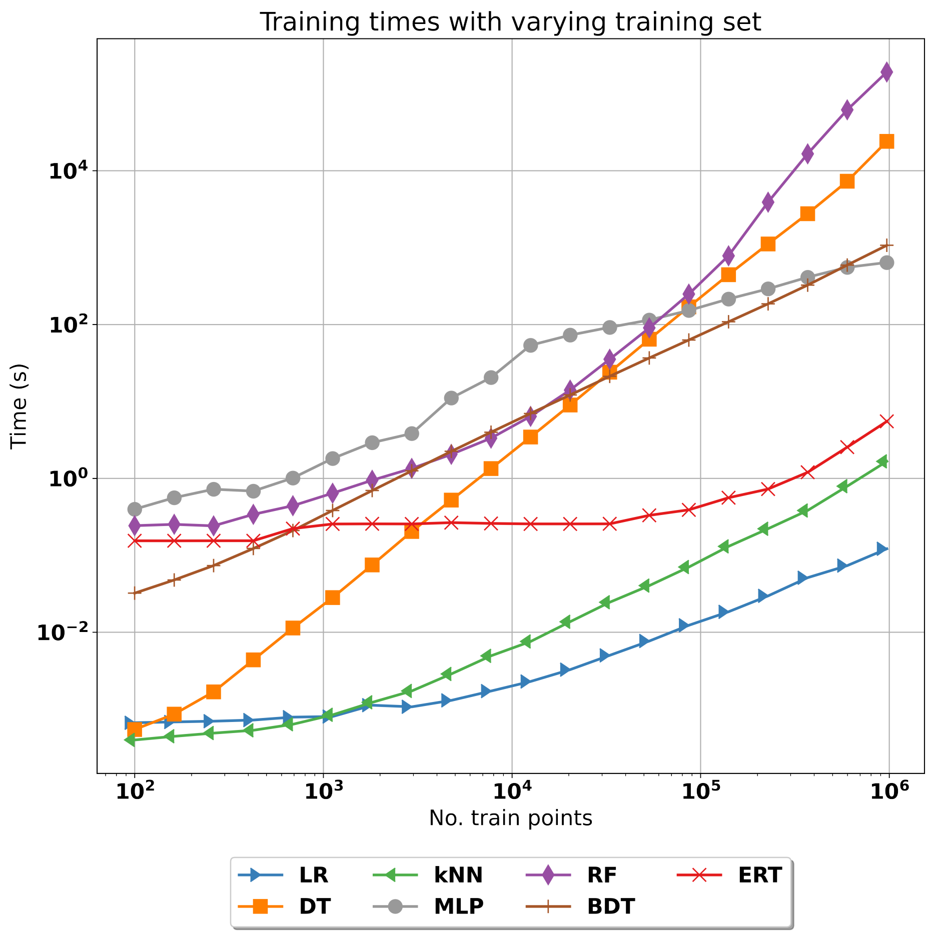

Figure 3.1: Graph of the training time plotted against the number of galaxies used in the training set to show how each algorithm scales with different sizes of training datasets. We saw that the simpler LR, *k*NN, and DT algorithms all begin as the fastest to train, however, the DT had terrible scaling and for large training sets became one of the slowest algorithms. Conversely the ERT and MLP algorithms began as two of the slowest algorithm to train, but scaled much better than the rest and could be more useful for massive training datasets.

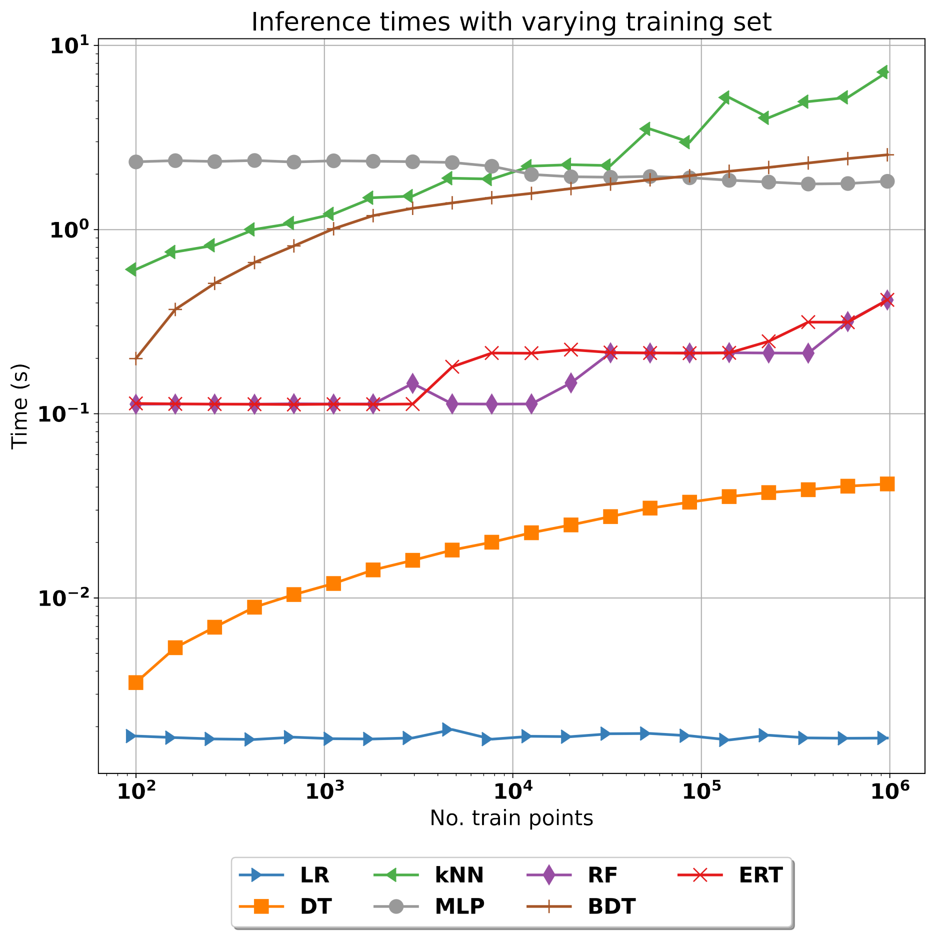

Figure 3.2: Graph of the inference time plotted against the number of galaxies used in the training set to show how each algorithm scaled with different sizes of training datasets (and a constant test set of 327870 galaxies). We saw all algorithms other than LR and MLP exhibit a training bloat, whereby the inference time increased with the number of galaxies included in the training set; however, the algorithms inference times generally increased by only a factor of 10 despite the training dataset increasing by a factor of 10^4.

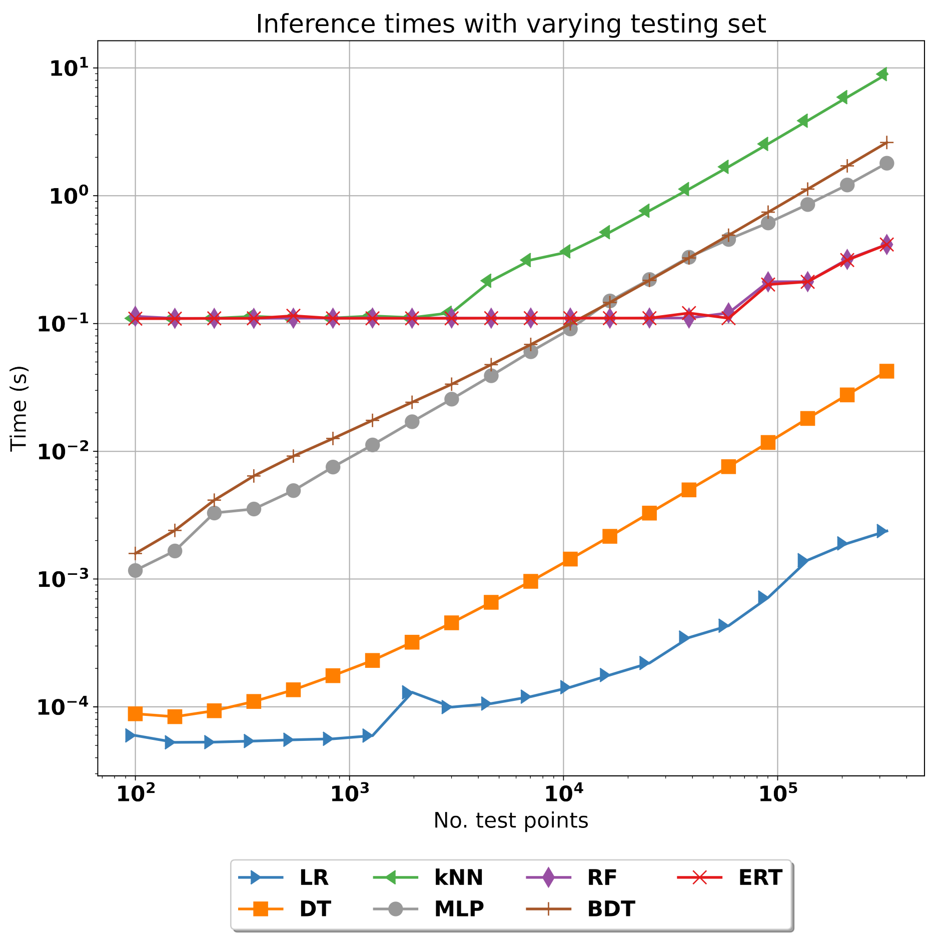

Figure 3.3: Graphs of the inference time plotted against the number of galaxies used in the testing set to show how each algorithm will scale with different sizes of testing datasets (and a constant training set of 983608 galaxies). In inference we saw all algorithms scaling very similarly with the main difference being the RF and ERT where, during the period between 10^2 to 10^5 galaxies used in the test set, the inference time didn't increase despite the number of galaxies to provide an estimate for increasing by a factor of 10^3. This meant that both algorithms ended up being faster to provide redshift estimations for larger test sets.

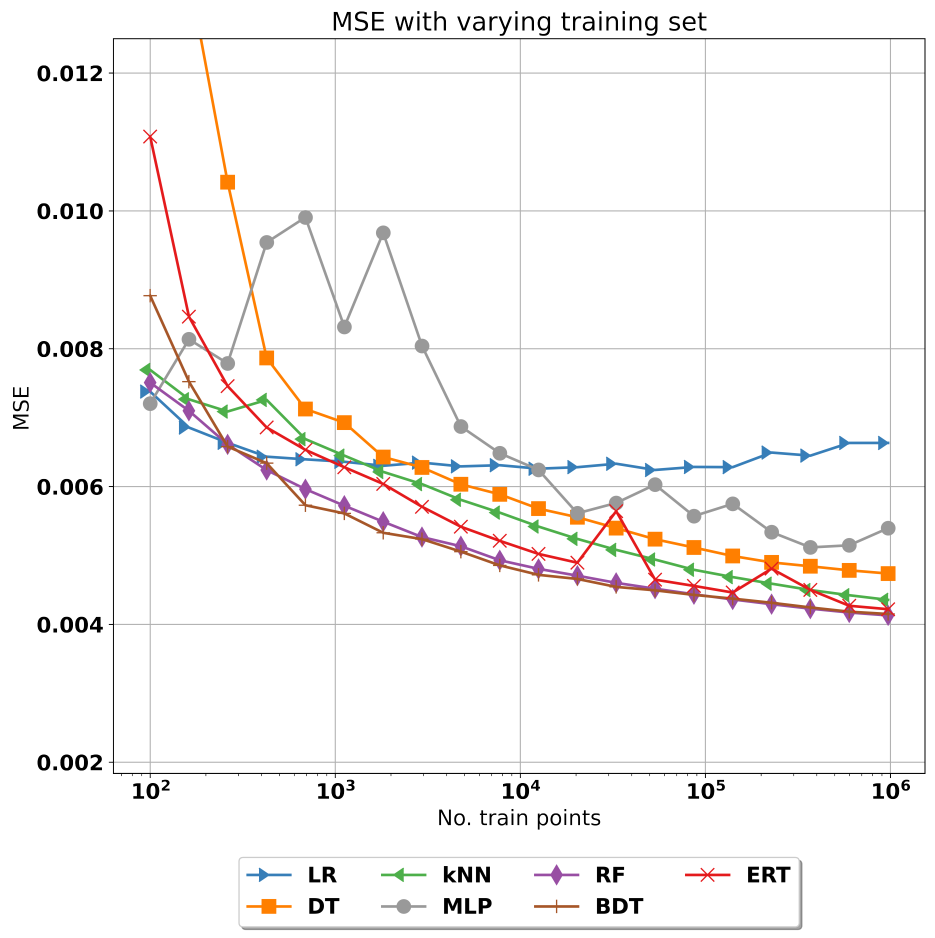

Figure 3.4: Graphs of the Mean Squared Error (MSE) plotted against the number of galaxies used in the training set to show how each algorithm's performance will scale with different sizes of training datasets (and a constant test set of 327870 galaxies). As expected, in general we saw all algorithms (other than LR) achieving lower MSE as the number of galaxies included in the training set was increased. However, we saw this increased error performance quickly plateau, and past 10^4 galaxies in the training set there was relatively little reduction in error.

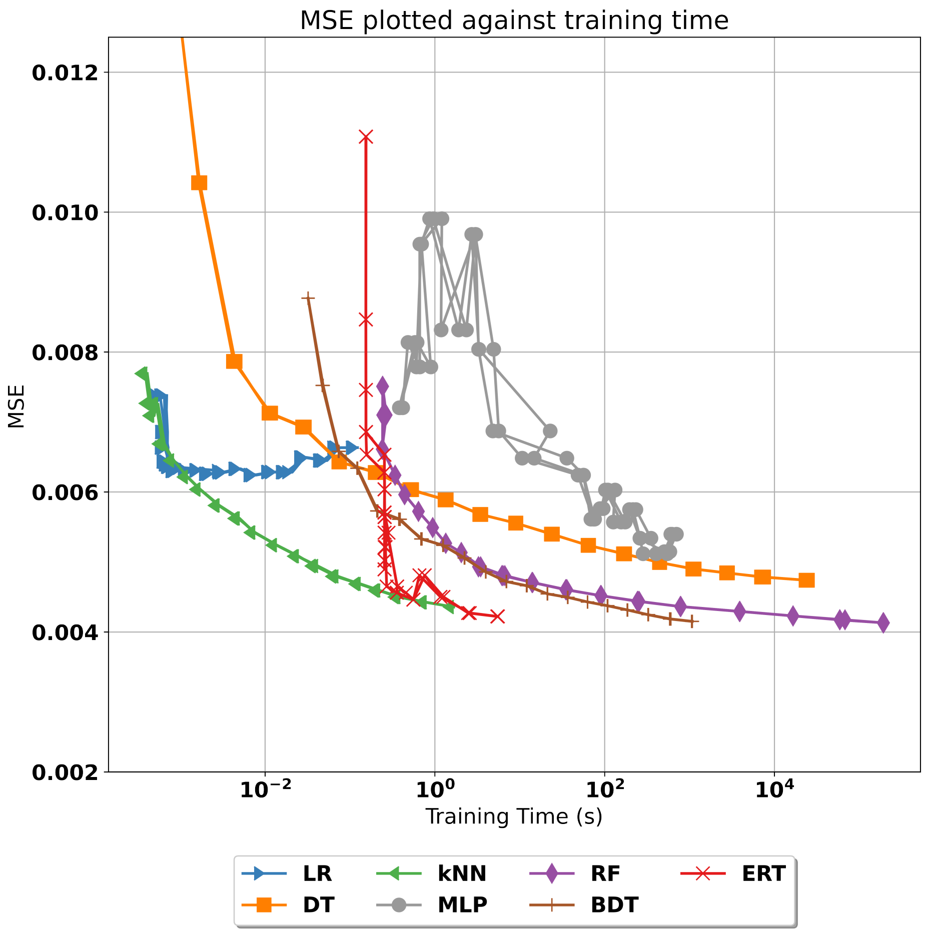

Figure 3.5: Graphs of the Mean Squared Error (MSE) plotted against the training times for each algorithm tested. For the ideal algorithm one would see a curve down to the bottom left corner, thereby achieving the best possible result for the error in the shortest amount of time. In general the algorithms do improve their errors as the training time (and number of galaxies included in the training set) increased; however, as shown, the improvement and time taken varies greatly for each algorithm. The RF achieved the lowest error but with the longest training time, whereas the ERT can be seen to reach a very similar error much faster.

3.4 Results

The results given in Table 3.2 show how the seven machine learning algorithms performed at producing photometric redshift estimations. Furthermore figure 3.6 displays the true spectroscopic redshifts plotted against the photometric redshift estimates for each machine learning algorithm. We also plotted the distributions of the redshift estimations for each of the algorithms as well as the true spectroscopic redshift in a violin plot in figure 3.7 to quickly see which algorithms were able to capture the correct distribution.

From these results we saw that all algorithms were able to successfully provide photometric redshift estimations. Using the violin plots from figure 3.7 we could see that the rough distribution was recovered by each algorithm, with the Multi-layer Perceptron (MLP) producing a slightly more similar shape to the true redshifts. However, from simply looking at the outputs shown in figures 3.6 & 3.7 it would be very difficult to determine which algorithm would be best to use. While the Decision Tree (DT) might be excluded due to its estimates being put into bands at set redshifts, its errors were still found to be quite low and it outperformed both the Linear Regression (LR) and MLP algorithms which had the worst metrics, with LR generally overestimating the redshift and MLP underestimating it.

Looking at the metrics in Table 3.2 alone, the Random Forest (RF) performed best having the lowest errors with a mean absolute error (MAE)$= 0.0385$ and mean squared error (MSE)$= 0.0042$, however, the other algorithms k-Nearest Neighbours (kNN), Boosted Decision Tree (BDT), and Extremely Randomised Trees (ERT) all performed incredibly similarly with MAE< 0.042 and MSE< 0.0046. Indeed, kNN had the lowest bias of all algorithms, although, this seemed to be due its over predictions and under predictions balancing as it also had a higher scatter and outlier fraction than the other algorithms. However; with such close performances of all these algorithms, it was impossible to sufficiently determine which algorithm would be the most useful. To be able to further differentiate between them and determine which would be the best algorithm to use, it was therefore necessary to use the benchmarking results.

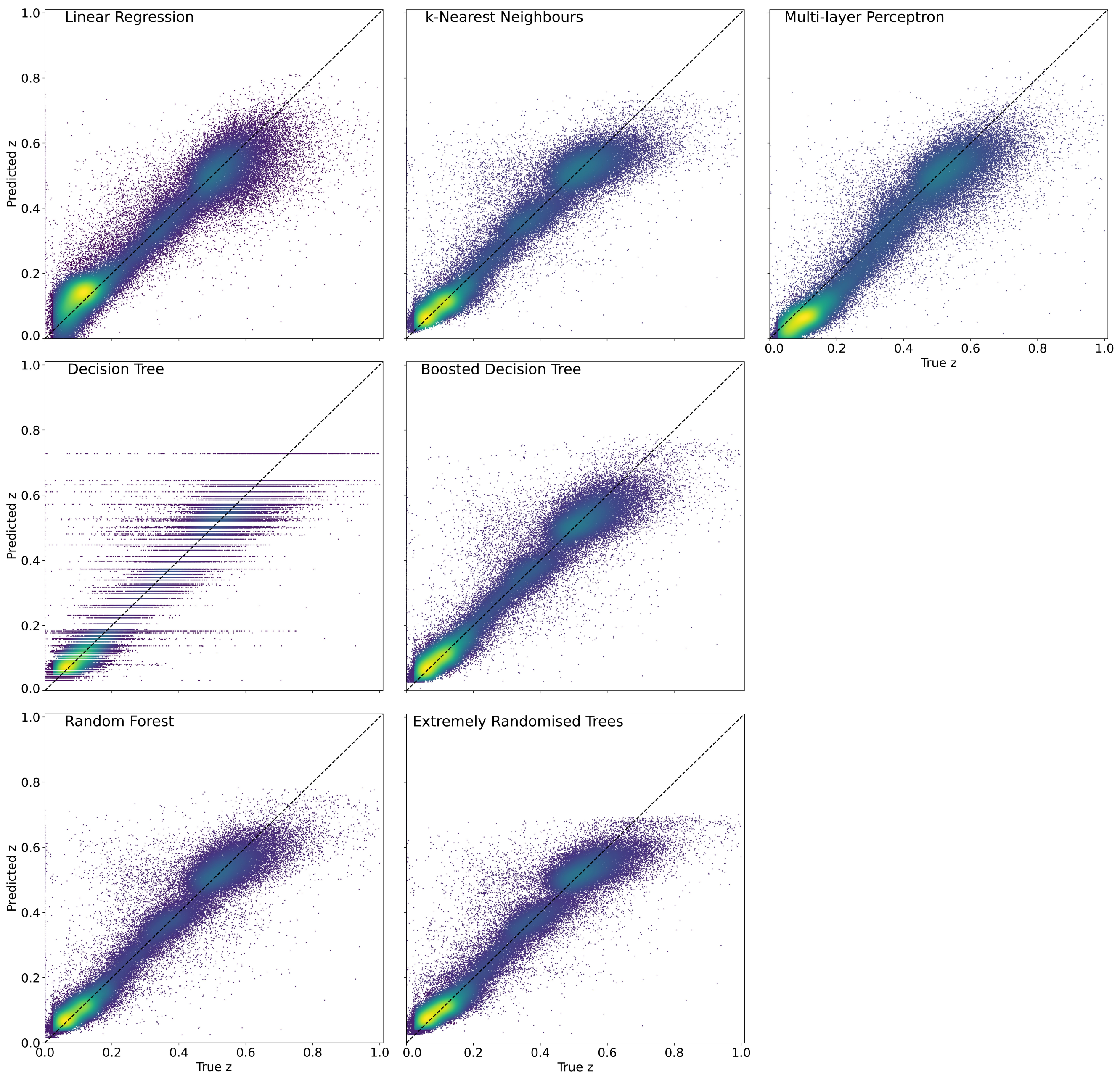

Figure 3.6: Graphs of photometric redshift estimates against the true spectroscopic redshift where the lighter shaded contours display the more densely populated regions. From top left to bottom right - Linear Regression (LR), *k*-Nearest Neighbours (*k*NN), Multi-layer Perceptron (MLP), Decision Tree (DT), Boosted Decision Tree (BDT), Random Forest (RF), and Extremely Randomised Trees (ERT).

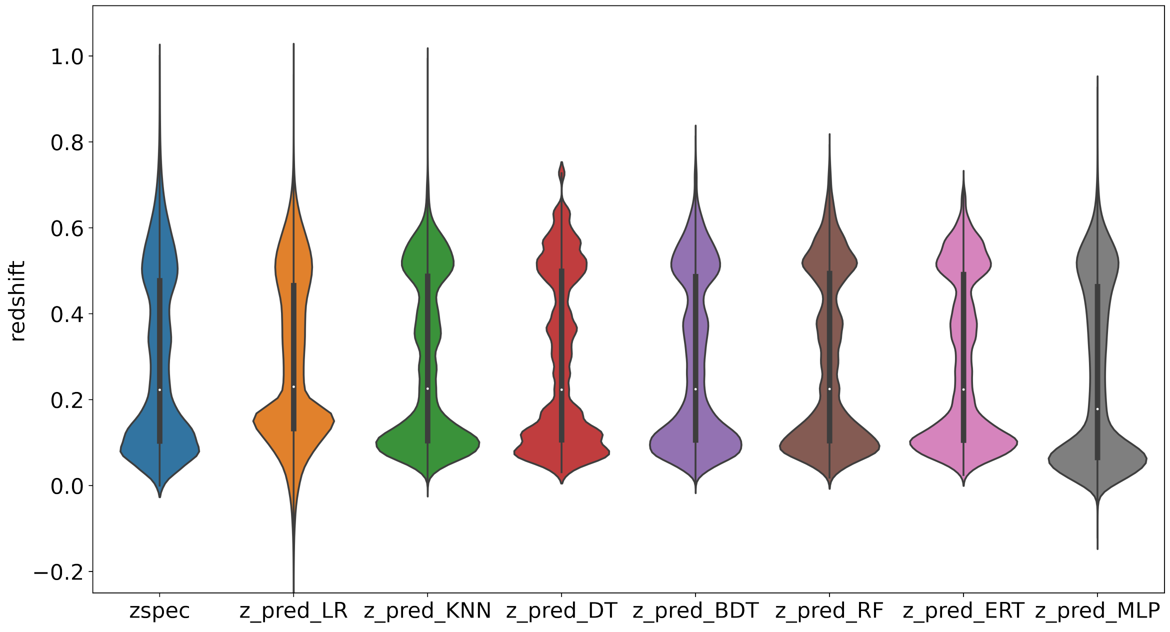

Figure 3.7: Violin plots showing the kernel density estimation of the underlying distributions of photometric redshift estimates of each algorithm along with the true spectroscopic redshift. From left to right - True spectroscopic redshift (zspec), Linear Regression (LR), *k*-Nearest Neighbours (*k*NN), Decision Tree (DT), Boosted Decision Tree (BDT), Random Forest (RF), Extremely Randomised Trees (ERT), and Multi-layer Perceptron (MLP).

The results of the benchmarking performed for each algorithm are plotted in figure 3.1 (that shows the speed of training with varying sizes of training datasets), figures 3.2 & 3.3 (that show the inference speeds with varying sizes of either training or testing datasets), figure 3.4 (that shows how the MSE varies as the number of galaxies in the training set increases), and figure 3.5 (that shows how the MSE varies with the time taken during training).

From these figures it was clear that increasing the number of galaxies in the training set resulted in a near linear increase in the training times (as would be expected), and there was a similar linear relationship between the numbers of galaxies in the test set and the inference times. The exception was the *k*NN, RF and ERT algorithms which began with a constant inference time irrespective of the number of galaxies in the test set. This was presumably due to the nature of the algorithms storing data to make their inferences, which also resulted in much slower inference times to begin with.

The scaling of the inference time with the number of galaxies in the training set shown in figure 3.2 also displayed how parametric algorithms (with a fixed number of parameters e.g. LR and MLP) had constant inference times regardless of the number of galaxies in the training set, whereas the remaining algorithms all experienced a 'training bloat' whereby the inference time increased as the number of galaxies in the training set was increased.

As shown by these benchmarking figures, the fastest algorithm overall was LR which remained the fastest both in training and inference with increasing sizes of training and testing datasets. This was perhaps not surprising as out of the algorithms tested it was the most simple model and as such required less computational resources both to train the model and to make its predictions. However, as LR also had by far the worst errors out of the algorithms tested (with errors around 30% higher than those of the better performing algorithms), it seemed unlikely that it would ever be implemented for the problem of photometric redshift estimation.

Out of the other algorithms the DT and MLP were the poorer performing in terms of error. The DT was the second fastest behind LR in terms of inference, using its simple decision rules to quickly obtain the redshift estimations; however, as it also resulted in only estimating certain redshift bands, the final estimates weren't as useful as other algorithms. Furthermore, the DT was the worst scaling algorithm for training and became the second slowest algorithm to train on a million galaxies. The MLP was also one of the slowest algorithms tested, starting as the slowest to train with small training sets and also being one of the slowest in inference. Although it's the simplest example of deep learning, it suffered from being one of the more complex algorithms tested, and would perform better on even larger datasets with far more features, where it would have more chance to catch up to the other algorithms in both speed and error performance.

The remaining *k*NN, BDT, RF, and ERT algorithms all performed well in terms of error and were the hardest to differentiate between, however, using the benchmarking results it was possible to see how differently they scaled. *k*NN was the simplest of the four better performing algorithms, and using the nearest neighbours to produce its estimates resulted in the second fastest training times, only being beaten by LR. Although *k*NN was very fast to train, it was the slowest in inference and exhibited a bad training bloat. While most other algorithms also displayed some level of this training bloat, it was worst for *k*NN due to the nature of its nearest neighbour search which became more and more computationally expensive as more training points were added, and as such it wouldn't be as useful an algorithm for giving estimates for incredibly large datasets.

Out of the three ensemble tree-based methods, the RF scaled the worst in terms of training, becoming the slowest algorithm to train on the 1 million galaxies. Whereas, the ERT scaled surprisingly well and became the third fastest algorithm in training and similar to *k*NN. In training the BDT was quite fast, scaling much better than the RF but worse than the ERT; however, when it came to inference the BDT scaled worse than both the ERT and RF and was the second slowest algorithm for large datasets. The RF and ERT scaled almost identically in inference, which made sense being such similar algorithms, both only being beaten by the much simpler LR and DT.

As a result it seemed that there was no clear best performing algorithm, but rather each algorithm could be useful in different situations. While the RF had the best error metrics, its terrible scaling with increasing training data meant that it would only be the best algorithm for problems where it could be trained once and it would be inefficient to use for problems which required the algorithm to be regularly retrained on large amounts of data. In that case the BDT which had similar errors but was faster to train could be a more useful alternative, and similarly if both the training and inference times were required to be lower the ERT would be a good compromise.

3.5 Conclusions

Producing reliable photometric redshift estimations will continue to be an incredibly important area of cosmology, and with future surveys producing more data than ever before it will be vital to ensure that the methods chosen to produce the redshifts can be run efficiently.

Here we showed how benchmarking can be used to provide a more complete view of how various machine learning algorithms scale with differing sizes of training and testing datasets. By combining the benchmarking results and regression metrics, we were able to demonstrate how it's possible to distinguish between algorithms which appear to perform almost identically and suggest which could be better to implement in different scenarios. Furthermore, by suggesting a novel time-considered optimisation process which takes into account the benchmarking results during model selection, it was possible to provide additional insight into how machine learning algorithms can be fine-tuned to provide more appropriate models.

From our tests we determined that while the kNN, BDT, RF, and ERT methods all seemed to perform very similarly, obtaining a good result for the MSE < 0.0046, it was the RF which achieved the best metrics, and was also one of the faster algorithms in inference. However, depending on which area of the pipeline an experiment requires to be faster, the RF method could also be inefficient as it scaled worse than all other algorithms in training. Hence for problems which require regular retraining of models on large datasets one of the other algorithms such as the BDT or ERT could allow for a greater improvement. As large sky surveys producing enormous datasets will require the most efficient methods possible it could also be necessary to investigate the use of deep learning neural networks which could benefit the most when using even larger amounts of data with more features.

Further work could be done to include a wider range of machine learning algorithms, including deep learning networks, and to test them on larger simulated datasets to confirm their scaling. By making use of time-considered optimisation it would also be possible to further examine the trade-offs between minimising errors and the training/inference times in each individual algorithm. We could also run the benchmarks on a variety of computer architectures, making use of GPUs which have the potential to speed up the algorithms that are most parallisable, as well as allowing us to examine the environmental impact of running such computationally expensive tasks.

Chapter 4

Deep learning for photometric redshift estimation using image data

This chapter is based on work which at the time of writing was being prepared for publication in the Monthly Notices of the Royal Astronomical Society (MNRAS).

Knowing the redshift of galaxies is one of the first requirements of many cosmological experiments, and as it's impossible to perform spectroscopy for every galaxy being observed, photometric redshift (photo-z) estimations are still of particular interest. Here, we investigate different deep learning methods for obtaining photo-z estimates from images, comparing these with 'traditional' machine learning algorithms which make use of magnitudes retrieved through photometry. As well as testing a convolutional neural network (CNN) and inception-module CNN, we introduce a novel mixed-input model which allows for both images and photometric data to be used in the same model as a way of further improving the estimated redshifts. We also perform benchmarking as a way of demonstrating the performance and scalability of the different algorithms. The data used in the study comes entirely from the Sloan Digital Sky Survey (SDSS) from which 1 million galaxies were used, each having 5-filter (ugriz) images with complete photometry and a spectroscopic redshift which was taken as the ground truth. The mixed-input inception CNN achieved a mean squared error $(MSE) = 0.009$, which was a significant improvement (30%) over the traditional Random Forest (RF), and the model performed even better at lower redshifts achieving a $MSE = 0.0007$ (a 50% improvement over the RF) in the range of $z < 0.3$. This method could be hugely beneficial to upcoming surveys such as the Vera C. Rubin Observatory's Legacy Survey of Space and Time (LSST) which will require vast numbers of photo-z estimates produced as quickly and accurately as possible.

4.1 Introduction

In the past decade the number of galaxies observed by large sky surveys has been rapidly increasing with hundreds of millions of galaxies being imaged (Alam et al., 2015, Drlica-Wagner et al., 2018, De Jong et al., 2015). This ever growing number is set to rise even faster with upcoming surveys such as the Vera C. Rubin Observatory's Legacy Survey of Space and Time (LSST) (Tyson et al., 2003) and the Roman Space Telescope (formerly WFIRST - the Wide-Field Infrared Survey Telescope) (Spergel et al., 2015) which will observe many billions of objects. For most cosmological studies in which galaxies are used the redshift is a key property that is required; however, despite spectroscopic surveys such as the Dark Energy Spectroscopic Instrument (DESI) (Martini et al., 2018), only a very small fraction of galaxies have an associated spectroscopic redshift. Instead, photometric redshift (photo-z) estimations are necessary.

These photo-z estimates can be obtained using two different methods (or a combination of both): template fitting, or machine learning (ML). The template fitting method uses templates of the spectral energy distribution (SED), and by fitting the observed SED of the galaxy to the template, its redshift can be inferred (Bolzonella et al., 2000). ML methods instead use a large training set of galaxies with labelled, 'true' values for the redshift (the spectroscopic redshift) and learn a mapping from the features of the galaxy data to their redshifts (Collister & Lahav, 2004). For traditional ML techniques, such as Random Forests (RF) or k-Nearest Neighbours (kNN), these features are taken from the photometry, giving magnitudes in different filters which can be combined into colours. However, for deep learning methods the image itself can be used as the input, with the pixel values being akin to features (Hoyle, 2016).

There are benefits to each method with template fitting being an inherently physical model, and by making use of SED templates which are full spectra, they can be shifted to any redshift and allow for redshift estimates to be obtained in ranges without large spectroscopic datasets (Benitez, 2000). ML methods on the other hand require a representative training sample in the same redshift distribution as the targets and as a result are only valid in that range. Despite this limitation, ML has been increasingly implemented as a faster method which is able to produce very accurate photo-z estimates where there is a sufficiently large training set (Abdalla et al., 2011).

Recently great strides have been made in the field of deep learning, aided by improving computer architectures and faster graphics processing units (GPUs), far more difficult tasks can now be performed using much larger datasets (Kirk, 2007). In industry many companies have been taking advantage of these methods for tasks which range from creating outfits for fast-fashion (Bettaney et al., 2019), to self driving vehicles (Bojarski et al., 2016). In astronomy surveys such as the Dark Energy Survey (DES) (DES Collaboration, 2016), the Kilo-Degree Survey (KiDS) (de Jong et al., 2013), Euclid (Amendola et al., 2018), Hyper Suprime-Cam (HSC) (Aihara et al., 2018), LSST (Ivezić et al., 2019), the Roman Space Telescope (Spergel et al., 2015), and the Square Kilometer Array (SKA) (Dewdney et al., 2009), will be producing petabytes of data and machine learning provides a viable solution to the otherwise unimaginable task of data analysis on such a scale.

In addition to the difficult task of processing data on these scales, the photometric redshifts produced must also be highly accurate. Many cosmological analyses, such as weak lensing driven cosmology (Heymans et al., 2021, DES Collaboration, 2021), rely on having extremely low errors in the redshift estimates to allow for more confident predictions of the cosmological parameters (Hildebrandt et al., 2021, Myles et al., 2021). Indeed, LSST have stated in their science requirements that the error on the mean redshift must be below $0.003(1+z)$ (Mandelbaum et al., 2018). Greater accuracy may naturally be achieved through larger, or novel deep learning methods, but also through increased emphasis on model interpretability and robustness to input error.

There are many other benefits to being able to directly use images rather than photometric features for photo-z estimation, predominantly that the deep learning algorithms could extract far more information than from the magnitudes alone (Pasquet et al., 2019, Schuldt et al., 2020). Indeed previous studies such as Soo et al.

(2017) worked hard to include morphological parameters which are implicitly contained in the images and that the deep learning algorithms could extract. Furthermore, other work has been done which found that deep learning methods have been able to produce photo-z estimates which outperform the previously best performing traditional methods based on kNN or RFs (D'Isanto & Polsterer, 2018). However, the deep learning algorithms are often much slower and more computationally expensive to run, and there has been little investigation into whether the benefits of these methods is worthwhile in the long term.

Other studies have also demonstrated methods for estimating probability density functions (PDFs) which can be desirable for cosmological analyses. Cavuoti et al. (2017) introduced a binned method where the input data-point is modulated, and the relative frequencies of binned output estimates converted to probability densities. Pasquet et al. (2019) showed that a classifier can be used in a similar manner where each output ”class” is treated as part of a binned redshift distribution, while the softmax of the output is taken to represent a probability density. We treated this problem of photo-z estimation as a regression problem, where the models produced a point estimate for the redshift. This allowed us to demonstrate how the novel mixed-input methods performed and act as a proof of concept, as well as simplifying the comparison between the deep learning algorithms and traditional methods.

Here we investigate if it is worth using deep learning on images compared to traditional ML using only the magnitudes as features, applying different types of convolutional neural networks (CNNs) as well as mixed-input models which use both images and magnitudes as inputs. In Sec. 4.2 we describe the data collected and used to train and test the machine learning algorithms which are outlined in Sec. 4.3 along with the metrics and optimisation process. We then present the results in Sec. 4.4 with discussions about how the mixed-input inception CNN was able to achieve such low errors and outperform all the other algorithms, before concluding in Sec. 4.5.

4.2 Data

The data used to train and test the different machine learning algorithms came entirely from the Sloan Digital Sky Survey data release 12 (SDSS DR-12) (York et al., 2000, Alam et al., 2015). In this work, we compiled 1059678 data-points each representing a galaxy. We downloaded the Petrosian magnitudes and spectroscopic redshifts, then acquired their corresponding images, each one comprising the five wavelength bands (u, g, r, i, z). It was a requirement to have spectroscopy performed to be able to use the spectroscopic redshift as the ground-truth, and in order to directly compare methods, the photometric magnitudes were also required to use as features for both the traditional ML algorithms and mixed-input models.

While the total number of galaxies which met the requirement of having an associated spectroscopic redshift was closer to 2 million (Beck et al., 2016), we decided that a training set of 1 million galaxies was sufficient. This decision was made as there wouldn't be a large difference in error performance going from 1 to 2 million galaxies. Furthermore, the scaling of the algorithms would only be visibly different with changing orders of magnitude of the training set (as we display later in figures 4.7 - 4.9), and therefore simulations would have then been required to be able to include more galaxies and reach the next order of magnitude.

The dataset used was also kept clean by requiring complete photometry where there were no missing values of any magnitudes which could have negatively impacted the redshift estimations and biased the results. Furthermore, the redshift range was kept to only include galaxies with $z < 1$, with the final distribution of galaxies used shown in figure 4.1. Although this simplified the problem rather than having a larger redshift range, this distribution matched that of the overall SDSS spectroscopic sample as shown by Beck et al. (2016), and therefore allowed for valid estimates to be made for SDSS galaxies in this range. However, as there were far more galaxies without spectroscopic data, the dataset used wouldn't be representative of the entire SDSS survey and further generalisation would be needed to provide reliable estimates for other datasets.

To generate the images used by the deep learning algorithms we first down-

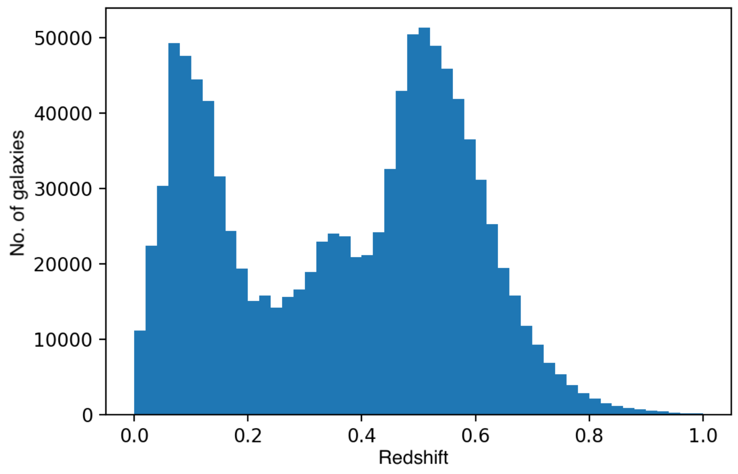

Figure 4.1: Plot showing the redshift distribution of the 1 million galaxies used in the study. The histogram was plotted with 50 redshift bins between $z = 0$ and $z = 1$ and displays the overall redshift distribution of galaxies in the SDSS spectroscopic sample with the two peaks caused by the difference between the galaxies observed during the main galaxy survey and BOSS as described by Beck et al. (2016).

loaded the full frames made available by SDSS which each comprised of five flexible image transport system (fits) files for the five different filters. For each galaxy within the frame we then generated 32×32 pixel images by centering the frame on the galaxy and cropping. We chose 32×32 pixel images as they were found to be sufficiently large to contain the full galaxy and surrounding sky for even the closer galaxies. When compared with 64×64 pixel images which other studies had previously used, the smaller galaxy images allowed for much smaller file sizes and more efficient computations as well as decreasing the incidence of image contamination by other galaxies. Tests were conducted using even smaller 20×20 pixel images, however, they resulted in worse performances by the CNNs, and as a result the 32×32 images were used throughout.

The final result was a set of five 32×32 pixel images for each galaxy which we saved in a numpy array (Harris et al., 2020) with shape $(32 \times 32 \times 5)$ which could then be used with the deep learning python package Keras (Chollet et al., 2015), our chosen interface for TensorFlow (Abadi et al., 2015).

4.3 Methods

With the magnitude data and images downloaded and in a usable format we were then able to apply the machine learning algorithms. The algorithms tested were as follows. A convolutional neural network (CNN) and an inception module CNN which both used the image data as the input. A random forest (RF) and extremely randomised trees (ERT) that had previously been found to be the best performing traditional methods (Henghes et al., 2021) and which only used magnitude features. And two experimental mixed-input models, which combined a CNN or inception module CNN with a multi-layer perceptron to use both the image data and magnitude features as inputs. Each of the individual algorithms is described in general in section 1.4 with further information about the specific models tested given below.

The algorithms were trained multiple times while varying the size of the training set used (up to 1 million galaxies), and tested on a fixed test set of 59678 galaxies. By doing this we were able to benchmark the different algorithms to determine their scalability as described in Sec. 4.3.7. The optimisation process is also described in Sec. 4.3.6 and the metrics used to evaluate the models' performances are given in Sec. 4.3.5.

4.3.1 Convolutional Neural Network

The full CNN architecture used in this study is shown in figure 4.2 and made use of two convolutional layers, each followed by an average pooling layer, to extract the information from the images. It then fed the flattened 4096 long array into two dense layers, with 1024 followed by 32 neurons, before a final dense layer was used to give the single value output for the predicted redshift. Rather than using the standard ReLU activation function (as defined in equation 1.35), an adapted parametric ReLU (PReLU) activation function, defined as

$$f(x) = \begin{cases} ax & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0, \end{cases} \tag{4.1}$$

was used where an additional parameter - the coefficient of leakage, a, was also learned during the training.

This resulted in the architecture that was found to be the most effective, simple CNN, and while deeper models were also tested, we found that the best performance occurred with only a few layers.

4.3.2 Inception Module CNN

Many variations of CNNs are possible, one example is the inception module CNN which makes use of inception modules as described in section 1.4.1.10. In the inception module CNN implemented here (shown in figure 4.4), we used two different sets of inception modules. The first had the same architecture as displayed in figure 1.15, however, after four sets of these inception modules the resulting output was too small to usefully apply a (5×5) kernel to, and instead we simply removed this layer from the inception module. Following this smaller inception module, the flattened 192 long array was fed into two further dense layers each with 1096 neurons before the final dense layer gave the predicted redshift.

Unlike the simple CNNs tested, when testing different configurations of inception module CNNs we found that the deeper models performed best. As well as changing the depth of the network we also tested different inception modules, changing the size of the kernels and swapping the max pooling layer for an average pooling layer. This resulted in a somewhat different model to that used by Pasquet et al. (2019) while still resembling the original GoogLeNet network.

4.3.3 Mixed-input Models

It is also possible to include multiple inputs in a CNN. This works by defining separate input data and running through networks in parallel before then combining the networks by concatenating similar to how the inception modules are built. In this work we experimented with two mixed-input models which took both SDSS images with shape $(32 \times 32 \times 5)$ used by the CNNs, as well as the 5 magnitudes (u, g, r, i, z) which were used as the sole features for the random forests. To handle the magnitude features a simple multi-layer perceptron (MLP) was used which included five fully connected layers each with 1024 neurons (for full network architectures used see figures 4.3 & 4.5).

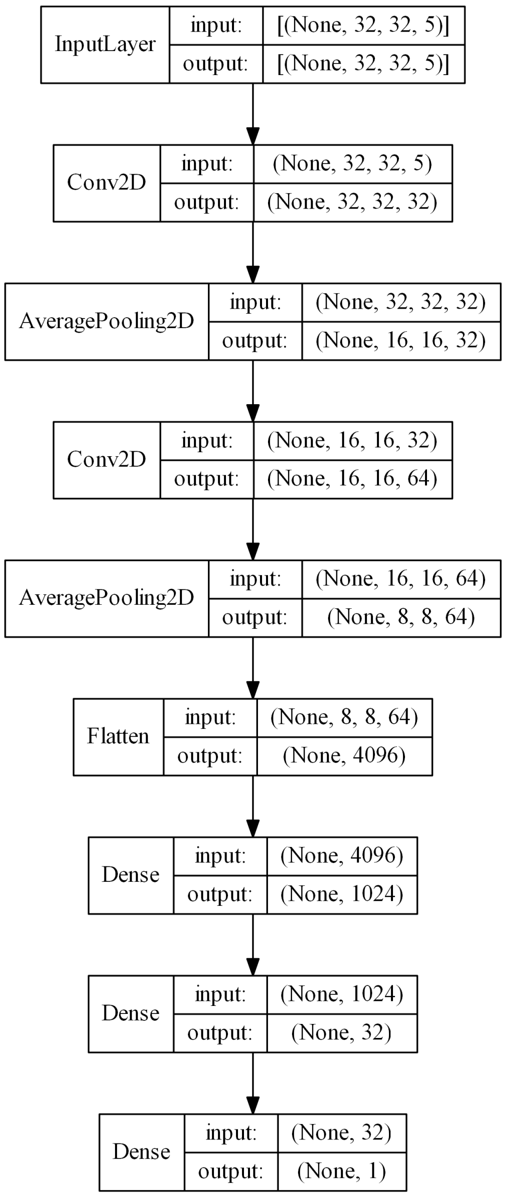

Figure 4.2: Network architecture of the base CNN tested. The CNN was constructed with two convolutional layers, each followed by an average pooling layer to reduce the dimensionality, before the feature map was flattened to give a 1D feature vector. This could then be handed to the two dense layers (which are the fully connected layers) that process the features before the final, single neuron layer is used to give the value of the predicted redshift.

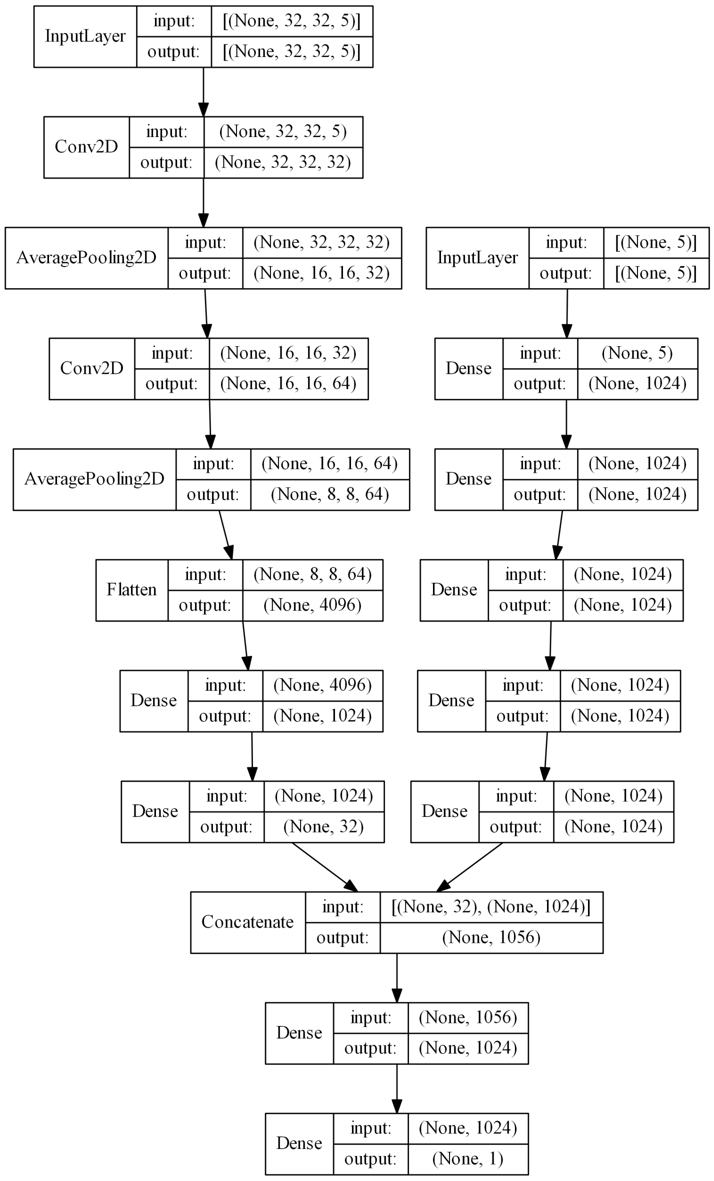

Figure 4.3: Network architecture of the mixed-input CNN. This model used the same CNN as 4.2 to handle the images, and added a MLP with 5 fully connected layers each of 1024 neurons to handle the magnitude data. The outputs of both were then concatenated before being handed to a fully connected layer and finally the single neuron layer which gave the value of the predicted redshift.

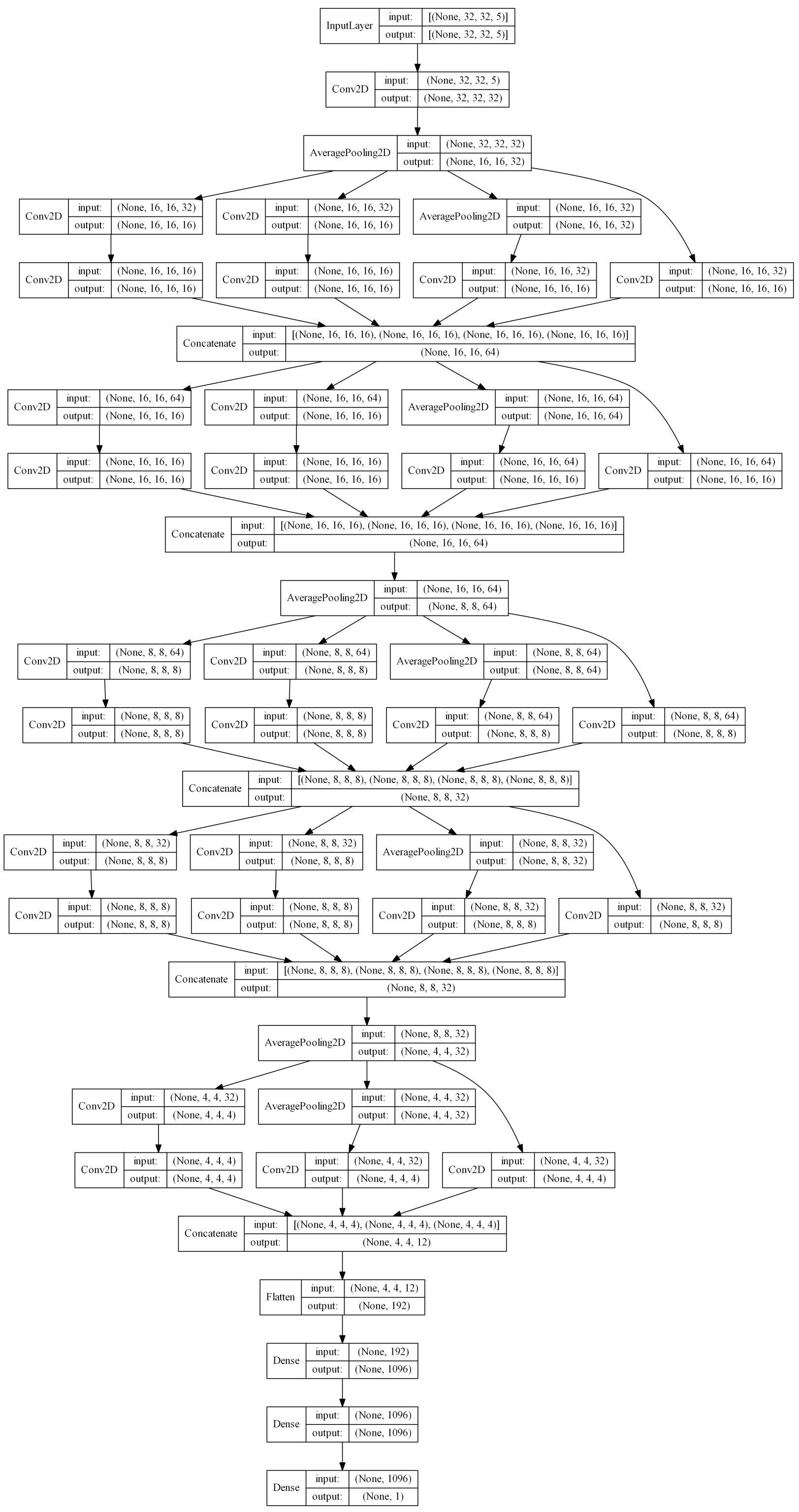
InputLayer input: [(None, 32, 32, 5)] output: [(None, 32, 32, 5)]
Conv2D input: (None, 32, 32, 5) output: (None, 32, 32, 32)
AveragePooling2D input: (None, 32, 32, 32) output: (None, 16, 16, 32)
Conv2D input: (None, 16, 16, 32) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 32) output: (None, 16, 16, 16)
AveragePooling2D input: (None, 16, 16, 32) output: (None, 16, 16, 32)
Conv2D input: (None, 16, 16, 16) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 16) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 32) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 32) output: (None, 16, 16, 16)
Concatenate input: [(None, 16, 16, 16), (None, 16, 16, 16), (None, 16, 16, 16), (None, 16, 16, 16)] output: (None, 16, 16, 64)
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 16)
AveragePooling2D input: (None, 16, 16, 64) output: (None, 16, 16, 64)
Conv2D input: (None, 16, 16, 16) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 16) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 16)
Concatenate input: [(None, 16, 16, 16), (None, 16, 16, 16), (None, 16, 16, 16), (None, 16, 16, 16)] output: (None, 16, 16, 64)
AveragePooling2D input: (None, 16, 16, 64) output: (None, 8, 8, 64)
Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 8)
AveragePooling2D input: (None, 8, 8, 64) output: (None, 8, 8, 64)
Conv2D input: (None, 8, 8, 8) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 8) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 8)
Concatenate input: [(None, 8, 8, 8), (None, 8, 8, 8), (None, 8, 8, 8), (None, 8, 8, 8)] output: (None, 8, 8, 32)
Conv2D input: (None, 8, 8, 32) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 32) output: (None, 8, 8, 8)
AveragePooling2D input: (None, 8, 8, 32) output: (None, 8, 8, 32)
Conv2D input: (None, 8, 8, 8) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 8) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 32) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 32) output: (None, 8, 8, 8)
Concatenate input: [(None, 8, 8, 8), (None, 8, 8, 8), (None, 8, 8, 8), (None, 8, 8, 8)] output: (None, 8, 8, 32)
AveragePooling2D input: (None, 8, 8, 32) output: (None, 4, 4, 32)
Conv2D input: (None, 4, 4, 32) output: (None, 4, 4, 4)
AveragePooling2D input: (None, 4, 4, 32) output: (None, 4, 4, 32)
Conv2D input: (None, 4, 4, 4) output: (None, 4, 4, 4)
Conv2D input: (None, 4, 4, 32) output: (None, 4, 4, 4)
Conv2D input: (None, 4, 4, 32) output: (None, 4, 4, 4)
Concatenate input: [(None, 4, 4, 4), (None, 4, 4, 4), (None, 4, 4, 4)] output: (None, 4, 4, 12)
Flatten input: (None, 4, 4, 12) output: (None, 192)
Dense input: (None, 192) output: (None, 1096)
Dense input: (None, 1096) output: (None, 1096)
Dense input: (None, 1096) output: (None, 1)

Figure 4.4: Network architecture of the inception module CNN. This model used a single convolutional layer and average pooling layer before applying 5 inception modules, where the fifth inception module was a modified version to be smaller and not include a (5×5) kernel. Following the inception modules the output was flattened to give the feature vector which was processed by two fully connected layers with 1096 neurons, and finally the single neuron layer to give the predicted redshift.

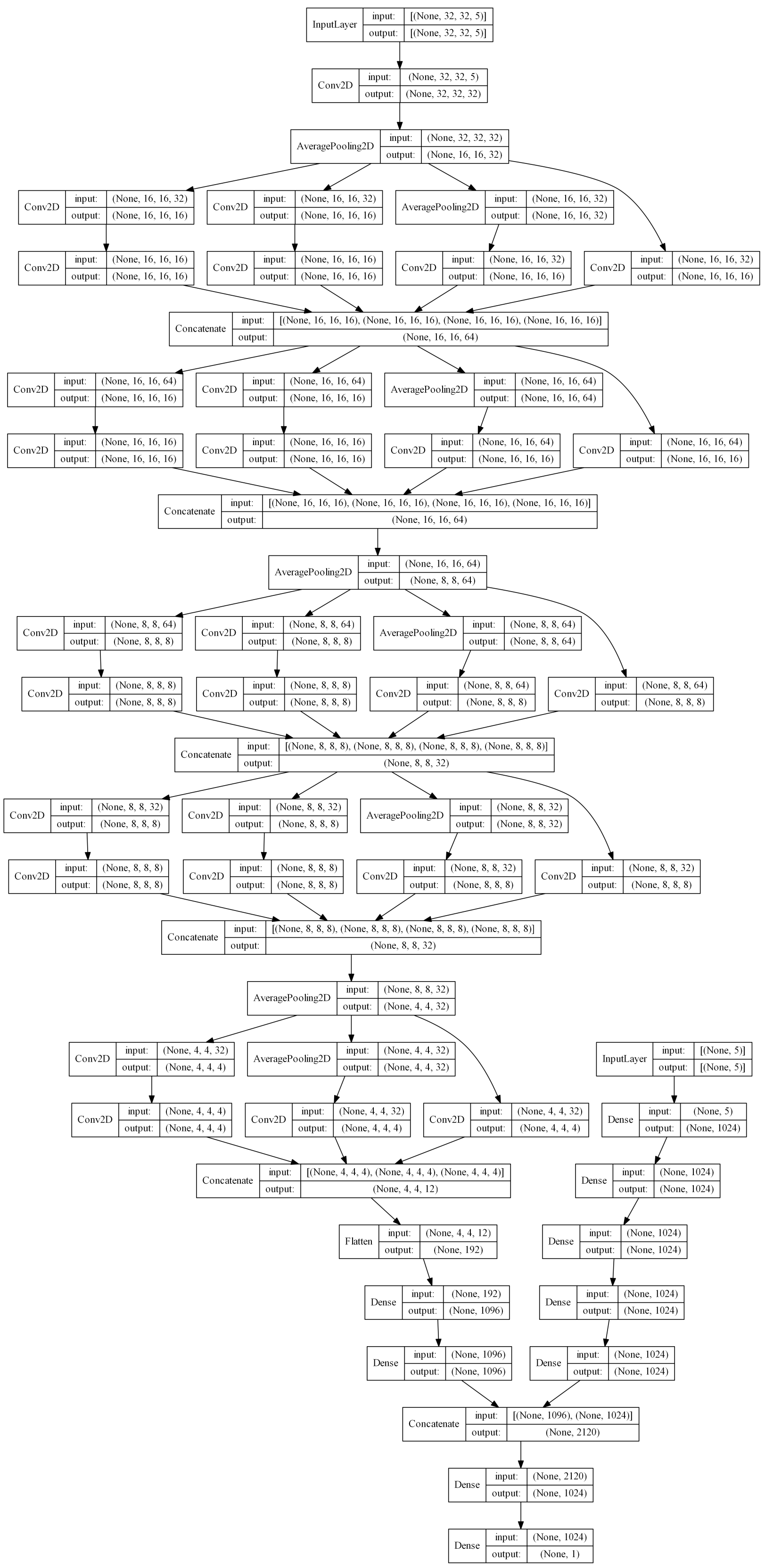
InputLayer input: [(None, 32, 32, 5)] output: [(None, 32, 32, 5)]
Conv2D input: (None, 32, 32, 5) output: (None, 32, 32, 32)
AveragePooling2D input: (None, 32, 32, 32) output: (None, 16, 16, 32)
Conv2D input: (None, 16, 16, 32) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 32) output: (None, 16, 16, 16)
AveragePooling2D input: (None, 16, 16, 32) output: (None, 16, 16, 32)
Conv2D input: (None, 16, 16, 16) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 16) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 32) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 32) output: (None, 16, 16, 16)
Concatenate input: [(None, 16, 16, 16), (None, 16, 16, 16), (None, 16, 16, 16), (None, 16, 16, 16)] output: (None, 16, 16, 64)
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 16)
AveragePooling2D input: (None, 16, 16, 64) output: (None, 16, 16, 64)
Conv2D input: (None, 16, 16, 16) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 16) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 16)
Conv2D input: (None, 16, 16, 64) output: (None, 16, 16, 16)
Concatenate input: [(None, 16, 16, 16), (None, 16, 16, 16), (None, 16, 16, 16), (None, 16, 16, 16)] output: (None, 16, 16, 64)
AveragePooling2D input: (None, 16, 16, 64) output: (None, 8, 8, 64)
Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 8)
AveragePooling2D input: (None, 8, 8, 64) output: (None, 8, 8, 64)
Conv2D input: (None, 8, 8, 8) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 8) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 64) output: (None, 8, 8, 8)
Concatenate input: [(None, 8, 8, 8), (None, 8, 8, 8), (None, 8, 8, 8), (None, 8, 8, 8)] output: (None, 8, 8, 32)
Conv2D input: (None, 8, 8, 32) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 32) output: (None, 8, 8, 8)
AveragePooling2D input: (None, 8, 8, 32) output: (None, 8, 8, 32)
Conv2D input: (None, 8, 8, 8) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 8) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 32) output: (None, 8, 8, 8)
Conv2D input: (None, 8, 8, 32) output: (None, 8, 8, 8)
Concatenate input: [(None, 8, 8, 8), (None, 8, 8, 8), (None, 8, 8, 8), (None, 8, 8, 8)] output: (None, 8, 8, 32)
AveragePooling2D input: (None, 8, 8, 32) output: (None, 4, 4, 32)
Conv2D input: (None, 4, 4, 32) output: (None, 4, 4, 4)
AveragePooling2D input: (None, 4, 4, 32) output: (None, 4, 4, 32)
InputLayer input: [(None, 5)] output: [(None, 5)]
Conv2D input: (None, 4, 4, 4) output: (None, 4, 4, 4)
Conv2D input: (None, 4, 4, 32) output: (None, 4, 4, 4)
Conv2D input: (None, 4, 4, 32) output: (None, 4, 4, 4)
Dense input: (None, 5) output: (None, 1024)
Concatenate input: [(None, 4, 4, 4), (None, 4, 4, 4), (None, 4, 4, 4)] output: (None, 4, 4, 12)
Dense input: (None, 1024) output: (None, 1024)
Flatten input: (None, 4, 4, 12) output: (None, 192)
Dense input: (None, 1024) output: (None, 1024)
Dense input: (None, 192) output: (None, 1096)
Dense input: (None, 1024) output: (None, 1024)
Dense input: (None, 1096) output: (None, 1096)
Dense input: (None, 1024) output: (None, 1024)
Concatenate input: [(None, 1096), (None, 1024)] output: (None, 2120)
Dense input: (None, 2120) output: (None, 1024)
Dense input: (None, 1024) output: (None, 1)

Figure 4.5: Network architecture of the mixed-input inception CNN. This model used the same inception module CNN as 4.4 to handle the images, and added the same 5 layer MLP which was used in 4.3 to handle the magnitude features. The outputs of both were concatenated and handed to a single fully connected layer before the final single neuron layer gave the predicted redshift.

A perfect CNN would theoretically be able to extract all useful information from the galaxy images (including the magnitudes which are obtained from performing photometry on the same images), and hence render the mixed-input models superfluous. However, as we saw from the results, the mixed-input inception CNN was able to outperform the inception CNN it was based off where the only difference was the magnitude features also being included. This suggested that the CNN wasn't able to perfectly extract the magnitudes from the images and by explicitly providing them as additional features to be handled by a MLP we were able to boost the performance.

In other cases of machine learning it can be ill-advised to use features which are highly correlated. This is due to the chance that the model will output results which vary drastically and wouldn't generalise to other datasets (Goldstein, 1993). However, due to the difference in extracting information from the images through convolutional layers which result in more abstract features than the magnitudes, and the fact that the results we saw suggested the explicit inclusion of magnitude features helped rather than hurt the model performance, we concluded that this wasn't a multicollinearity issue (Garg & Tai, 2013).

There are two additional reasons for avoiding using correlated features. First, is an increase in model complexity which results in slower models than if fewer, independent features were used. However, in this case the inclusion of a MLP to handle the additional magnitude features had a minimal effect on the overall speed of the model (as can been seen in figures 4.7&4.8). The second reason is that using correlated features often makes models less interpretative. This point was also less of an issue in this case as the addition of the magnitudes as features was physically motivated and the MLP that handled these features was kept separate to the rest of the CNN.

This allows the mixed-input models to be thought of as a combination of two separate models. This process of taking a combination of different models to give the final prediction is more widely used, and when using neural networks one has the option to implement subnetworks (as we have done here). Subnetworks, such

Table 4.1: Grids of hyperparameters that were used in the RF and ERT, selected by the random optimisation.

Classifier	Hyperparameter	Selected Value
RF	"no. estimators"	**200**
	"max. features"	**2**
	"min. samples leaf"	**7**
	"min. samples split"	**3**
	"min weight fraction leaf"	**0**
	"criterion"	**mse**
ERT	"no. estimators"	**147**
	"max. features"	**4**
	"min. samples leaf"	**3**
	"min. samples split"	**87**
	"min weight fraction leaf"	**0**
	"criterion"	**mse**

as the CNN and MLP we used, handle the different inputs separately before their outputs are concatenated into a single feature vector which can then be used to give the final output prediction (Burkov, 2019).

4.3.4 Traditional Algorithms

As well as the deep learning models discussed above, two traditional machine learning algorithms were tested to compare the neural networks with machine learning methods which only used magnitude features. The two methods chosen were a Random Forest (RF) and Extremely Random Trees (ERT) algorithm which had previously been found to be the best performing traditional methods for the problem of photo-z estimation (Henghes et al., 2021). For details of these tree based algorithms implemented in this study (and described in section 1.4) we provide the hyperparameters used in table 4.1.

4.3.5 Metrics

One of the most important steps of any machine learning problem is defining sensible metrics which can be used to evaluate model performance. As we treated this as a regression problem, where we found a single value for the photometric redshift, the natural choice was to calculate the three most commonly used regression metrics: mean squared error (MSE), mean absolute error (MAE), and R squared score

(R^2).

As well as these standard regression metrics, there are three additional metrics which are most commonly used in photometric redshift estimation: bias - the mean of the residuals (defined in equation 3.2), precision (also $1.48\times$ median absolute deviation (MAD) as defined in equation 3.3) - which gave the expected scatter, and catastrophic outlier fraction - the fraction of predictions with an error greater than a set threshold, here set > 0.10 (as defined in equation 3.4). The results of our tests are given in table 4.2 with each of the metrics quoted for the various different algorithms.

4.3.6 Optimisation

Optimisation is the process of fine-tuning the hyperparameters of machine learning algorithms to give the best possible predictions. These hyperparameters are parameters that are set previous to the learning process and dictate how the algorithms create the mapping from input data to answer. For traditional methods, such as the RF and ERT, this optimisation can be done by specifying a grid of hyperparameters and iteratively testing which combination of parameters gives the best predictions. This process of testing every combination of the specified grid (brute-force optimisation) is very slow, and instead we tested 200 random combinations within the hyperparameter grid (in our random optimisation) which gave a good estimate of the best possible parameters in much less time.

Neural networks such as the CNN, inception module CNN, and mixed-input models cannot be optimised in the same way. As well as being far more computationally expensive which would make any brute-force optimisation impossibly slow, each network has a unique architecture which changes the hyperparameters that need optimising. In this study we tested various architectures, instead of defining grids of hyperparameters to test we simply went through the hyperparameters which have the greatest impact on the models (such as the number of layers, the number of neurons in each layer, and the kernel size, stride, and padding of convolutional layers) and tested different combinations to find the best preforming models.

4.3.7 Benchmarking

Benchmarking is the process of running a set of standardised tests to determine the relative performance of an object, in this case iteratively running the training and testing of different machine learning algorithms. Here, benchmarking was performed in a similar vein to chapter 3. We recorded the time taken throughout the machine learning process and varied the size of the training dataset to be able to compare the efficiency of the various models. By combining these measurements with the error results for the photometric redshifts we were able to better understand the performance of the different algorithms and give more discussions along with plots in section 4.4.

Generally the hardware used to test each algorithm should be kept the same, however, in this case as we were comparing CNNs with traditional ML methods, the CNNs were trained using a graphics processing unit (GPU) whereas the RF and ERT were trained using the central processing unit (CPU). While this did change the nature of the comparison, it was still a valid test as both the GPU and CPU used were simply standard laptop components (an Nvidia GTX1050Ti and intel i9-8950hk). Additionally this highlights one of the key differences between deep learning methods, which are highly parallelisable, and traditional ML methods, which often aren't. Even in the case of RFs which are also parallelisable, and indeed were run over multiple CPU cores, they don't benefit to the same extent as CNNs can when run using thousands of cuda cores (Kirk, 2007).

4.4 Results

The results of our investigation are presented in multiple ways. Table 4.2 details the error metrics achieved by each of the machine learning algorithms when using the full 1 million galaxies in the training set. We plotted density-scatter graphs of the predicted photo-z estimates against the true spectroscopic redshift in figure 4.6, as well as plotting the results of the benchmarks in figures 4.7-4.10 to show how the different algorithms scaled. Finally, the results of running the same algorithms on a smaller redshift range of $z < 0.3$ are discussed in section 4.4.1.

Table 4.2: Results of testing the different machine learning algorithms, where each algorithm was trained using 1000000 galaxies. The RF and ERT both used photometric data, whereas the CNN and Inception module CNN used images data, and the mixed-input CNNs used both images and photometry.

	Random Forest (RF)	Extremely Randomised Trees (ERT)	Convolutional Neural Network (CNN)	Inception Module CNN	Mixed-input CNN	Mixed-input Inception CNN
MSE	0.01253	0.01261	0.01009	0.00956	0.00997	**0.00916**
MAE	0.05003	0.05067	0.04388	0.04310	0.04154	**0.03966**
R^2	0.76154	0.76002	0.80809	0.81810	0.81030	**0.82567**
Bias	0.00498	0.00538	0.00122	$-$**0.00094**	0.00292	0.00878
Precision	0.03076	0.03170	0.02985	0.02987	0.02764	**0.02588**
Catastrophic Outlier Fraction	0.04722	0.04866	0.03619	0.03309	**0.03048**	0.03075

From these results we saw that the mixed-input inception module CNN was the best performing algorithm in terms of errors with a mean squared error $(MSE) = 0.009$. It also had the best performance in every other metric other than catastropic outlier fraction and bias, where the mixed-input (standard) CNN had a slightly better outlier fraction with both algorithms having just over 3% outliers, and the inception module CNN had a lower bias due to its remarkably symmetric predictions as shown in figure 4.6.

Figure 4.6 also displays a potential downfall in the majority of the CNN-based methods, with all but the standard CNN having an initial redshift cut below which they failed to predict any redshifts at all. While this was a result of both the architectures selected and the smaller number of galaxies included in the training set with a redshift $z < 0.05$, one would need to be careful to ensure that any algorithm implemented in a photo-z pipeline could predict redshifts in the full range.

While the mixed-input inception module CNN showed impressive perfor-

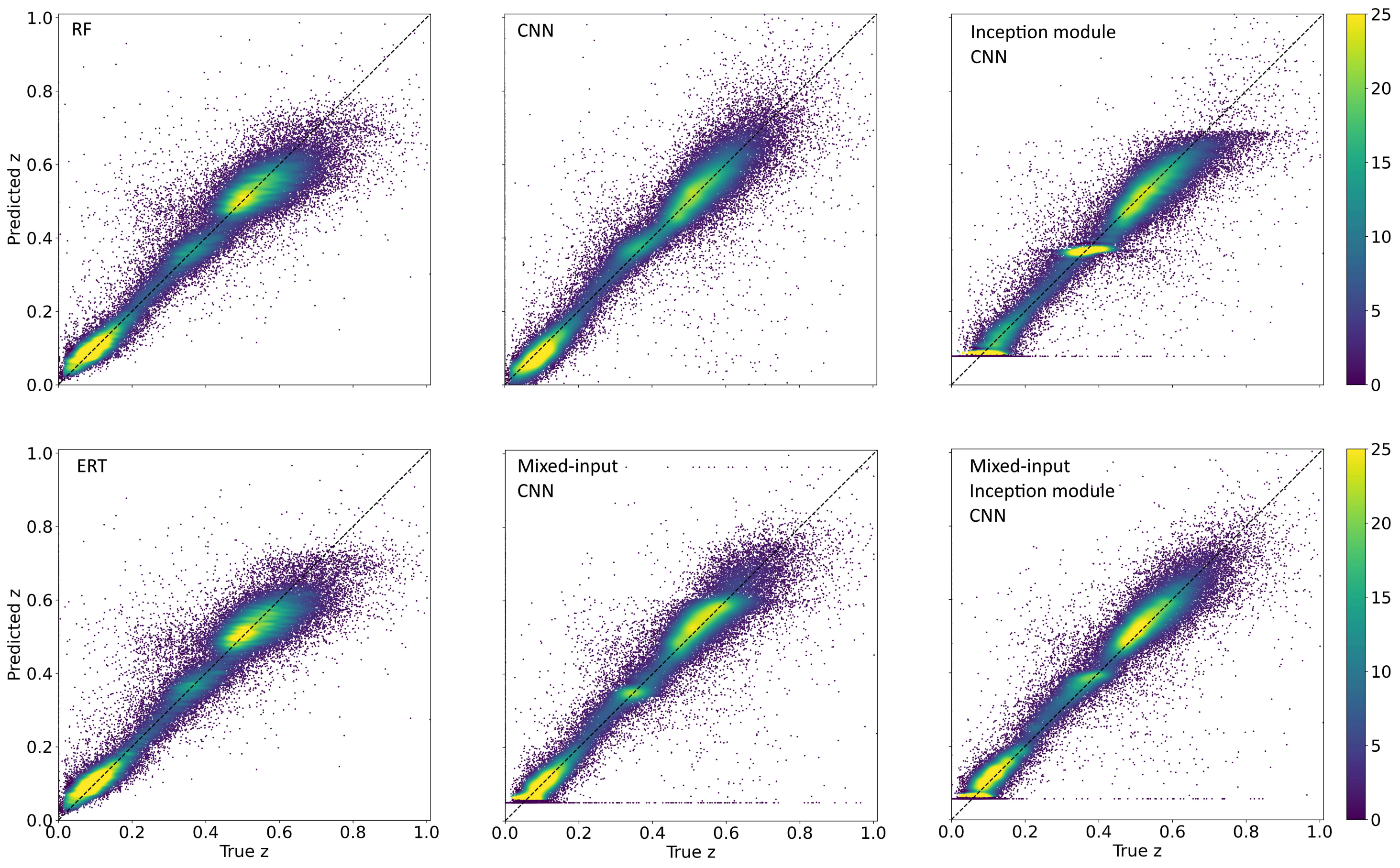

Figure 4.6: Density-scatter plots of the predicted photometric redshift estimates against the true spectroscopic redshift for each of the algorithms tested.

mance it did come at the cost of being the slowest algorithm tested, which made sense being the most complex model tested. As seen from figures 4.7 & 4.8 the two inception module based CNNs were the slowest algorithms, with the mixed-input model being only slightly slower in both training and inference than the image only model. Similarly the mixed-input CNN was only slightly slower compared with the CNN, showing that the addition of the magnitude features to the image based CNNs means only a small increase in computational requirements.

We also saw that the traditional machine learning methods, RF and ERT, were significantly faster but also worse in terms of their error performance. While these methods could still be very useful in the absence of image data, the improvements seen by making use of CNNs make them an exciting option. Furthermore, by di-

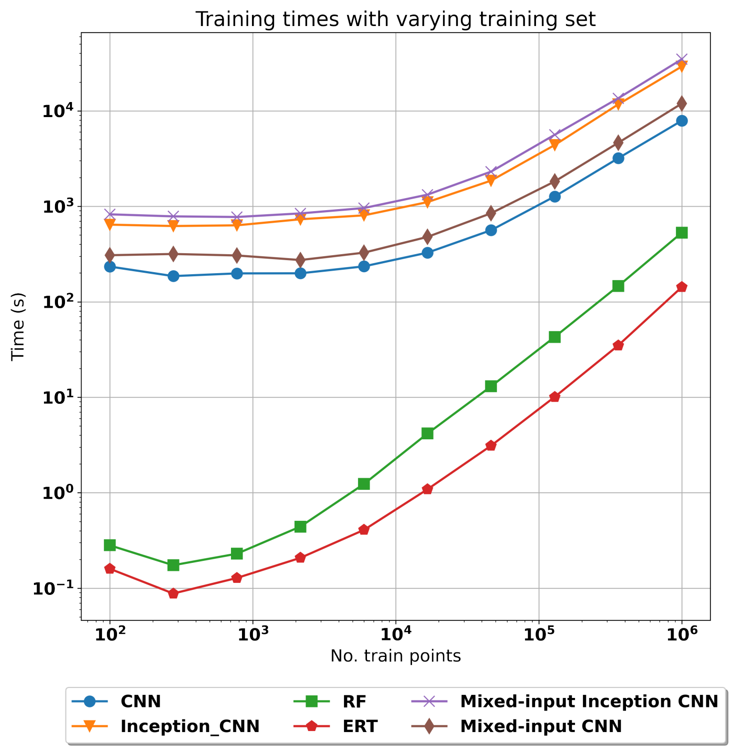

Figure 4.7: Plot showing how the training time changes with the number of galaxies used in the training set to display how each algorithm scaled.

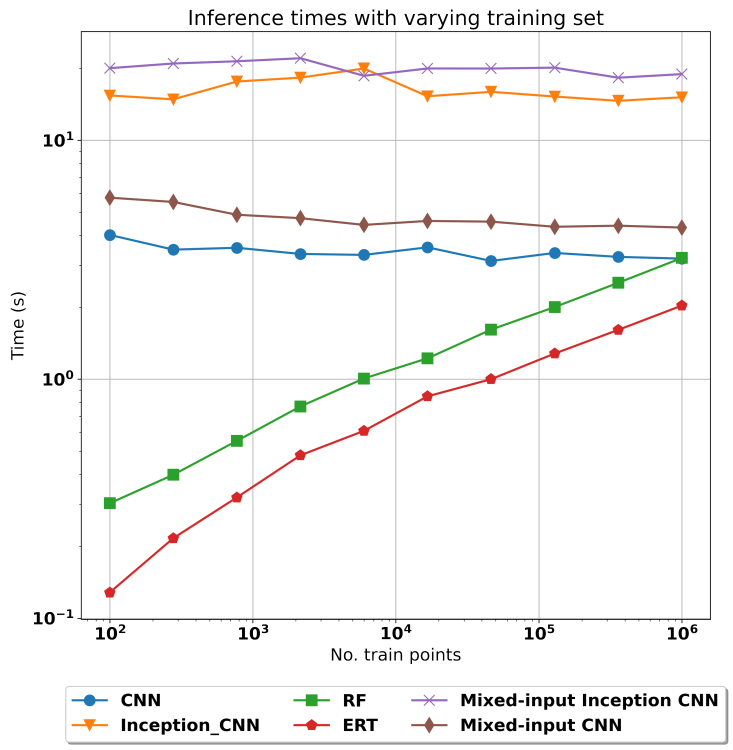

Figure 4.8: Plot showing how the inference time changes with the number of galaxies used in the training set to display how each algorithm scaled.

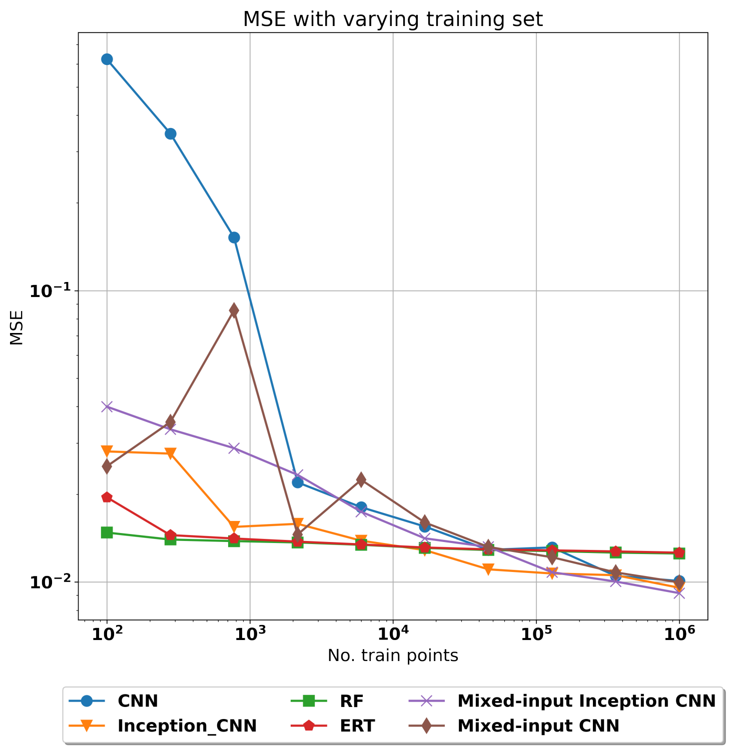

Figure 4.9: Graphs of the Mean Squared Error (MSE) plotted against the number of galaxies in the training set to show how each algorithm's performance scaled.

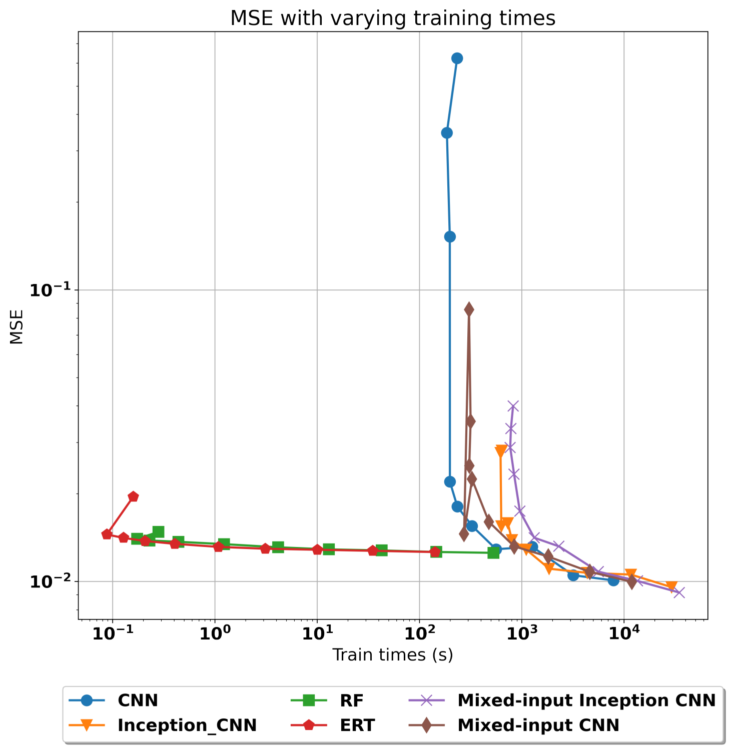

Figure 4.10: Graphs of the Mean Squared Error (MSE) plotted against the time taken to train the algorithms.

rectly using the image data rather than the magnitude features, one could offset the increased time required to train the algorithms with the time saved due to not needing to previously extract features.

What's more, the RF and ERT also showed the worst scaling of all the algorithms, slowing down at a faster rate as the number of galaxies included in the training set was increased. This was due to the fact that the tree-based algorithms are non-parametric and the complexity increases with the increasing dataset, whereas the neural networks have a fixed size. Past 1 million galaxies in the training set the RF was already slower in inference than the CNN, and with large enough datasets it's possible that they could become almost as slow during training. If for the largest datasets CNNs become faster than traditional methods then their main setback of being slower and more computationally expensive would no longer be of concern.

The performance of the RF and ERT highlighted the improvements possible when including image data rather than using magnitudes alone, with a reduction in errors of around 25%. The experimental mixed-input models also showed good potential to further improve performance, however, as the inception module CNN performed better than the mixed-input CNN, it was clear that the CNN network architecture had a greater impact than the addition of the magnitudes as extra features. The improvement from inception module CNN to the mixed-input inception CNN was also much less than the improvement from RF or ERT to the CNNs (the improvement from including images), with a further error reduction at just over 4%.

4.4.1 Lower redshift range

Although the algorithms performed well over the entire dataset we wanted to also test the performance for a smaller region to be able to more directly compare with other studies (such as Pasquet et al. (2019)) and see how much better the performance could be when the problem of estimating redshifts was made easier by only considering the range $z < 0.3$.

The exact same process was carried out using the same six algorithms algorithms and the results from the retrained machine learning algorithms are given in table 4.3. We also plotted the redshift estimations against the true spectroscopic

Table 4.3: Results of testing the different machine learning algorithms for the redshift range $z < 0.3$. Each algorithm was retrained using 400000 galaxies with the RF and ERT both using photometric data, whereas the CNN and Inception module CNN used images, and the mixed-input CNNs used both images and photometry.

	Random Forest (RF)	Extremely Randomised Trees (ERT)	Convolutional Neural Network (CNN)	Inception Module CNN	Mixed-input CNN	Mixed-input Inception CNN
MSE	0.00140	0.00162	0.00072	0.00070	0.00169	**0.00069**
MAE	0.02472	0.02855	0.01851	0.01773	0.02785	**0.01693**
R^2	0.76004	0.72196	0.87732	0.88074	0.71023	**0.88230**
Bias	0.00292	0.00128	0.00713	0.00447	−0.00670	**0.00070**
Precision	0.02218	0.02803	0.01875	0.01691	0.02631	**0.01543**
Catastrophic Outlier Fraction	0.02245	0.02393	**0.00557**	0.00659	0.02338	0.00816

redshift in figure 4.11.

From these we saw that in general the algorithms performed much better, reaffirming the fact that photometric redshift estimation becomes an easier problem over a shorter range. The mixed-input inception CNN continued to be the best performing algorithm with a $MSE = 0.0007$, however, there was far less separating the CNN and inception module CNN in the smaller redshift range. In fact, the CNN performed better than every other algorithm when it came to the catastrophic outlier fraction with only 0.56% outliers and one of the best constrained scatter plots (second only to the mixed-input inception module CNN).

Figure 4.11 shows how well constrained the redshift estimates of the mixed-input inception module CNN were, with a denser region along the $z_{\text{pred}} = z_{\text{spec}}$ line. Furthermore, the algorithm no longer exhibited its previous issue of not predicting redshifts across the full redshift range. Indeed the algorithms which had a redshift cut in the smaller redshift range were the RF, ERT, and mixed input CNN, however,

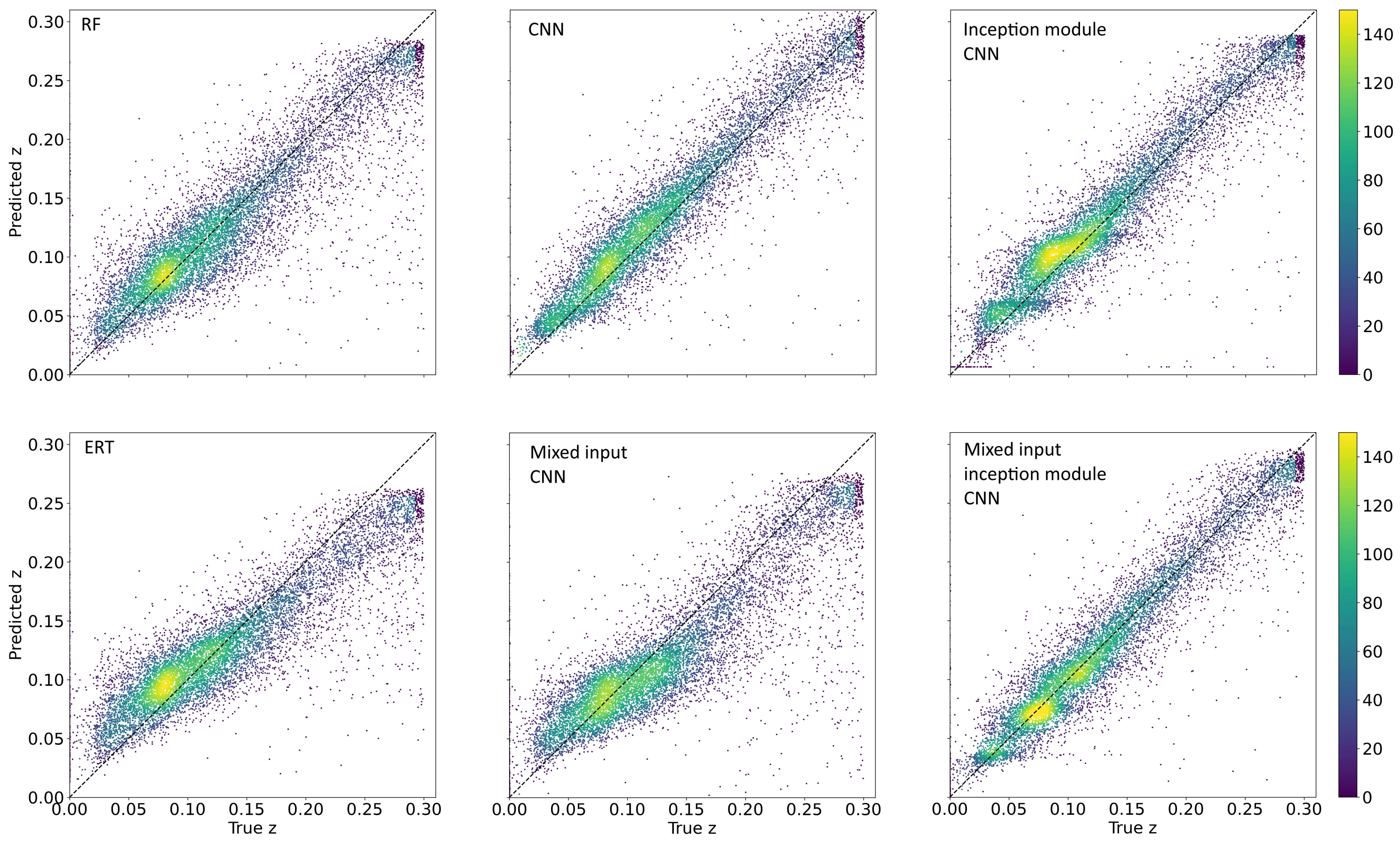

Figure 4.11: Graphs of the predicted photometric redshift estimates against the true spectroscopic redshift for the same machine learning algorithms retrained on galaxies within the range $z < 0.3$.

in this case they failed to predict redshifts above a certain value.

The only algorithm which didn't show the same improvements was the mixed-input (standard) CNN, which, when it came to the smaller redshift range, performed more similarly to the RF and ERT. As we saw the mixed-input inception model perform very well it seemed that there was still potential to make use of both images and photometry, however, by not optimising the algorithms for the new dataset and using the exact same network that was used for the entire redshift range, the mixed-input CNN clearly didn't transfer as well to the new data.

There was also a greater disparity between the other CNN based methods and traditional methods for this smaller redshift range. While there was around a 30%

improvement going from the RF and ERT to CNNs over the entire dataset, this increased to 50% for the smaller redshift range. The image based CNNs were therefore able to provide even more advantage in this range, suggesting that the additional information extracted from the images is even more beneficial in the smaller range, possible due to the fact that the galaxies would generally occupy a larger region of the image.

However, this boost in performance for the redshift range of $z < 0.3$ also highlighted a key failing of the algorithms, in that an ideal model would generalise well enough to perform just as well across the entire redshift range. This might not be realistic as by removing a large section of the data there was far less chance of having catastrophic outliers, and the overall problem was made easier.

4.5 Summary

Processing accurate photometric redshift estimations will remain a vital task of cosmological analyses. Future surveys such as LSST aim to observe more galaxies than ever before, and it is of utmost importance that the methods developed and implemented are both effective and efficient.

Here we have shown how image-based CNN methods compare to traditional tree-based methods which make use of magnitude features from photometry. We found that the additional information the CNNs were able to extract directly from the images of galaxies allowed for a significant reduction in errors. However, as the CNNs were more complex than the RF and ERT algorithms, they were also much slower to run and required far more computational resources.

Our results showed that the experimental mixed-input models in particular had great potential for photo-z estimation. Using 1 million images of galaxies to train the algorithm the mixed-input inception CNN was able to achieve a $MSE = 0.009$. Furthermore, when the problem was simplified to only include galaxies in the range $z < 0.3$, the model achieved an even more impressive $MSE = 0.0007$, outperforming the traditional RF by $> 50\%$.

Further work would include using even more data with tens or hundreds of millions of galaxies and images (which would require the use of large scale simulations and more powerful computer architectures). The use of more powerful CPUs and GPUs in high perfomance computing systems could allow for better practices in benchmarking and set a standard system. Additionally, by stretching the amount of data further we could then determine with certainty at what point the CNNs would become faster than the RF and ERT, as well as discover whether increasing the amount of data used in the training set would eventually have no effect on model performance. Finally, the models tested here could also be extended to produce PDFs for the estimated redshifts, and through further optimisation (including using custom loss functions) the errors could be reduced even more.

Chapter 5

Conclusions and Future Work

In this chapter I summarise the main results of the work described in chapters 2 - 4. I also outline some future work, which following on from the research of the previous chapters, could provide several more research projects for the future.

5.1 Thesis Summary and Future Work

5.1.1 Searching for TNOs in DES using Machine Learning

In chapter 2, I used machine learning to develop an additional step of a detection pipeline to help improve the efficiency of the search for distant Solar System objects in DES. I first tested eight different machine learning algorithms to determine which could make an effective classifier for the problem of detecting the rare objects in a very imbalanced dataset. Having determined the random forest classifier to perform best, I then further fine-tuned the algorithm to maximise the recall while still keeping the precision high enough to remove the majority of the noise.

Once fully optimised, the random forest was able to perform very well and achieved an AUC = 0.996, with a recall = 0.96 and precision = 0.80. This meant that when implemented in the DES TNO detection pipeline the random forest was able to remove 80% of the noise and run the orbit fitting stage over 5$\times$ faster, while keeping the vast majority of the possible detections. While a small number of possible detections would be missed by the algorithm (around 3.5%), as multiple detections are required to claim a discovery of a new object, there would be an even smaller chance of missing an object.

A natural extension to the work completed in section chapter 2 would be to test applying the same pre-processing stage to other objects to see if machine learning could as effectively be used to aid their detection. Closer TNOs would be the first starting point and instead of generating more extreme examples with possible planet 9s using the survey simulator, a wider population of TNOs could be generated to be used as the positive examples in the training set. As the rest of the detection pipeline is identical (where triplets are fitted to orbits to determine if they could result from real objects) the exact same analysis could be performed. The next step could then be to apply similar machine learning methods to help separate asteroid detections from noise.

As mentioned by the DES Collaboration (2016), DES is expected to have observed hundreds of thousands of asteroids, many of which are 'new' (not being listed by the minor planet centre). The difficulty arises in linking observations from the same night, as the asteroids are so close that their apparent motion is very high. One possibility could be to use deep learning methods which could inspect the entire difference-imaged field, or a section of the field, to classify objects which could be asteroids. This also raises the chance of using a similar method for TNOs, and instead of using the linked triplets, images could be used as the input.

5.1.2 Benchmarking ML methods for photo-z estimation

In chapter 3, I compared various machine learning algorithms for the task of obtaining photo-z estimates, and used this as an example of how benchmarking can be performed. In a similar vein to chapter 2, I tested the different algorithms to get an idea of how each performed, however, as a regression problem rather than classification the metrics and performance analysis was completely different, and the benchmarking allowed for more information about the scaling of the algorithms (up to a million galaxies) to determine which could be the most effective overall. Furthermore, during the optimisation process a new time considered optimisation was introduced which allowed for a compromise between the error and efficiency that could drastically speed up some methods with only a very small increase in error.

The random forest was again found to be the best performing algorithm in terms of the error metrics and achieved a MSE $= 0.0042$. Although the random forest gave the lowest errors, it was also one of the slowest methods and took much longer to train than other algorithms such as extremely randomised trees which also had very similar errors. This suggested that the best algorithm couldn't be determined by a single error metric and instead, depending on the situation where the algorithm was being implemented, the best algorithm to use would change. For instance, if regularly retraining the algorithm on new data, the extremely randomised trees algorithm would have been a better choice with much faster training times.

An extension of using deep learning methods directly with images is exactly what I did in chapter 4, however, there are several additional areas that could be explored in the future. First, when performing benchmarking of machine learning methods it would help to have a range of computer hardware which could be used to run the algorithms. By running the algorithms on both CPUs and GPUs, it could have been possible to determine the effects of the different computer architectures and quantify the improvement of using GPUs when implementing highly parallisable algorithms. What's more, a standard set-up could have been defined by which other experiments could measure themselves to, and by having an array of different machines, it would be easier for more experiments to directly compare their results.

Another natural augmentation to the work completed would be to undertake the same analysis using more data. Instead of showing the scaling of the algorithms up to 1 million galaxies, it would be valuable to be able to show the scaling for additional orders of magnitude (and reach the number of galaxies that large sky surveys will be able to observe). However, this raises the issue for supervised learning that we simply don't have real spectroscopic observations for that many galaxies. As an alternative to using real observations it could be possible to augment the existing dataset or use large simulations where the true redshift would be a known value set in the simulation.

5.1.3 Deep learning for photo-z estimation

In chapter 4, I expanded on the methods used in chapter 3 for photo-z estimation, this time including deep learning methods which could directly estimate the redshift from the galaxy images. As well as testing different CNN architectures, I compared these deep learning methods with the traditional random forest and extremely randomised trees algorithms which only took the magnitudes as features. Furthermore, I experimented with building mixed-input CNNs which could utilise both the image and magnitude data to see if a combination of methods would allow for an improved performance.

The deep learning algorithms, and specifically the inception module CNNs, were found to perform much better than traditional methods with errors 30% lower over the entire redshift range and 50% lower when only using galaxies with a redshift, $z < 0.3$. This suggested that the CNNs were able to extract more useful information by directly using the images compared with the traditional methods which only had the magnitudes as features. The tests of the mixed-input models suggested that it was possible to further boost performance, and the mixed-input inception module CNN was the algorithm with the best overall metrics, achieving a MSE = 0.009 over the full redshift range and MSE = 0.0007 for the range $z < 0.3$. By using the magnitudes as additional features on top of the images, the mixed-input inception CNN was able to estimate redshifts with a 4% lower error than the inception module CNN.

When using CNNs the process of optimising the networks is still quite ad hoc. Ideally this optimisation process would be formalised in a more similar way to optimisation processes of traditional machine learning algorithms. Future work could include designing a package to do this, where the optimisation would run through set network architectures depending on the type of problem and input data, before then testing a grid of hyperparameters such as the number of layers, the number of neurons in each layer, the kernel sizes, stride, and padding. This could be incredibly helpful for researchers across disciplines where CNNs have been implemented and allow for a more robust and explainable method of deciding what architecture to use.

In addition to this package, the time considered optimisation step which was introduced in chapter 3 could be formalised and expanded to also include CNNs. This could allow for an easier implementation of algorithms which are optimised not only to minimise error (or other metrics), but which also take into account the efficiency of the model. One of the major issues facing research making use of AI and machine learning is the ever increasing computational costs. These result in more and more computer clusters being required, increasing not just the monetary cost but also the environmental impact, and in many cases by using an optimisation which also takes time and efficiency into account, many computational (and hence environmental) resources could be saved.

5.2 Final words

Machine learning is one of the greatest tools at the disposal of astronomers, and while this thesis has focused on applying machine learning methods to different problems in astronomy, machine learning is also indispensable to researchers across various disciplines (with a few additional areas presented in the following appendices). However, it is also the responsibility of the researchers to ensure that they aren't reckless in their applications. Explainable models which are physically motivated must continue to be sought over any 'black-box' equivalent, and vital for the sustainability of the research, the models must also aim to be efficient. Following these stipulations, machine learning offers endless possibilities to the research community and will continue to be at the forefront of groundbreaking experiments for the foreseeable future.

Appendices: Machine Learning Beyond Astronomy

In the following three appendices I outline some additional work completed during my PhD which also helps highlight the interdisciplinarity of different machine learning methods and the benefits of applying them across different fields. As one of the inaugural students of University College London's (UCL's) Centre for Doctoral Training (CDT) in data intensive science, group projects were carried out in the first year with industry partners, and appendix A details the group project I completed at ASOS to create recommender systems. Another of the CDT's requirements was the completion of a six-month full time placement with one of the industry partners, and in appendix B I discuss the projects undertaken at ASOS during my placement. Finally, appendix C outlines a separate research project carried out during the Covid-19 pandemic in which I collaborated on an investigation into the effects of natural ultra-violet radiation on the rate of infections in the UK. More specifically, I performed the statistical analysis to find correlations between UV index and Covid-19 cases over different time periods.

Appendix A

An Investigation into Recommender Systems at ASOS

This appendix details the work completed for ASOS during the CDT's group project in the first year of the PhD. The aim of the group project was to develop a recommender system that could be used by ASOS to provide customised size recommendations. Simple models were tested to provide recommendations first for shoes, and later dresses, which were compared to a baseline model. It was found that a random forest classifier was able to slightly outperform the modal baseline reaching accuracies of around 75%; however, the improvements in recommendations were found to come more from strategic data management than the model implemented. Vitally, the purchases made by a single account first needed to be split into individual customers, as each account could represent multiple customers.

A.1 Introduction

Online fashion retailers have increasingly been taking a greater market share, in part due to their convenience which often includes a free returns policy. ASOS has been a champion of the hassle-free returns policy and it remains a key feature of their platform. The ability to try products with the knowledge of easily being able to return them without incurring a cost is a feature that drives sales, however, this results in a large percentage of purchases being returned. These returns are in part due to the buying habit of customers to purchase the same item in multiple sizes with the intention to return the worse fitting size. By providing customers with the best possible prediction of what correct size they should buy (the size which is most likely not to be returned), there could therefore be significant savings from reducing the fraction of items returned.

Recommender systems are a type of information filtering which take an input and aim to predict either a ranking or preference for a given user. They are commonly used in various areas including online video services, music streaming, advertising, and retail (Resnick & Varian, 1997). While there are many different ways of building a recommender system, typically two kinds of filters are used: collaborative filtering, and content-based filtering. Collaborative filtering works by examining the user's previous data (in this case shoe or dress sizes), and by also taking other users' data that are similar (have previously bought in the same sizes) a model learns to predict relevant items for the user. Content-based filtering uses the characteristics of the items themselves (e.g. the shoe/dress sizes and other properties such as the brand) and uses these to recommend items that are similar.

In this investigation we used both of these filtering methods by testing a random forest which used the sizes and brand information of items previously purchased to make predictions; as well as matrix factorisation which used both customer and item information mapped onto a joint space of latent features which could then be used to obtain recommendations. However, the models themselves were only one aspect of the investigation, and first the data had to be very carefully prepared. Difficulties could arise from the fact that many products which are labelled with the same size may actually vary in measurement. This is because there is no standard sizing system across different brands.

Another issue is vanity sizing, where manufacturers have a tendency to increase the measured size of their products compared to the label with the aim of encouraging sales (but which also acts to further vary the size disparity across brands). Furthermore, there are often simply errors in the manufacturing process which result in the measured size of the products being different to the stated measurement. Finally, the customers themselves add to the issue of finding a 'correct' size. Different people with identical measurements could have different preferences in terms of how their clothes should fit, and might not necessarily keep the same sized items.

For the purpose of this initial investigation, we began by considering only footwear since the size of a shoe corresponds more directly to a simple measurement of said shoe. Later tests were also conducted using dress data as a way of confirming the methods' validity and finding out how they performed on more complex data. In the following section I introduce the different methods implemented, starting with a simple modal baseline before testing a random forest and matrix factorisation. I then present the results in section A.3 and finally give a summary of the group project in A.4.

A.2 Recommeder Systems Tested

The first system tested was the simple modal method where the mode of the size of previous purchases was used as the prediction for future purchases. To begin, the most commonly purchased size for every male/female customer in 2016 was used to predict the size for every male/female customer in 2017. This gave a quick baseline for the minimum performance expected (as it tested the null model of predicting a single size for everyone on the platform). This baseline model was then improved in stages to start becoming a more intelligent recommender system. First, the data was filtered to only include customers who purchased items in both 2016 and 2017. This allowed for the second system to use the mode of sizes of each individual customer's purchases in 2016 to predict the size they purchased in 2017. However, while this is a vastly improved recommendation for the customers with previous purchases, it can't be applied to new customers.

The data was further filtered to use periods of 3 months instead of a year, using the mode of the size purchased in the previous 3 month period to predict the purchases made in the next months. This allowed for more data to be included as customers typically bought multiple items in shorter time periods. We further extended this by using random time periods, where any previous purchases could be used to predict future purchases. Other changes to the baseline then included rounding the sizes of shoes (as the half sizes were far less common), and finally, introducing account splitting.

Account splitting was the process of identifying accounts where their purchase history indicated multiple personas using the same account. We first identified the accounts with the most purchases, these weren't necessary being used by many people but rather could also be reselling items, either way the size recommendations were less relevant and allowed us to discount them. The easiest way to separate multiple customers in one account was by gender (where male products were purchased on a female account and vice versa). While some customers would purchase items designed for the opposite gender, for simplicity we assumed that we could split the account into two separate male and female customers (and this assumption seemed to be correct in the vast majority of cases). Finally, a clustering algorithm was applied to distinguish up to two customers for each gender, allowing for four possible customers in each account.

To build a smarter recommendation system, the first investigation was into using a supervised machine learning algorithm. The random forest was chosen over other algorithms on the basis that they can generally outperform other classifiers and are good for handling multiple classes. The features used by the random forest were simply the mode of the size previously purchased by a customer and the brand of the item purchased, and the label was taken as the size of the item the customer purchased and kept (assuming this to be their true size). This meant that the only real difference between the random forest and the baseline model was the extra brand information being used as a feature.

It was hoped that this extra information would allow the random forest to learn some relation between the brands and sizes and hence give improved predictions, and the initial tests were promising with slight increases in accuracy (of about 1%). When investigating the feature importance of the random forest, we found that the sizes had a much greater importance than the brand, with around a 90% importance for the mode size. This explained why the difference between the random forest and baseline was so little, as the predictions were being made from predominantly the same data and the model was able to learn that the modal method was a sensible solution.

The final method tested was matrix factorisation. We first constructed the training matrix with rows for each customer and columns representing the sizes. Each entry in the matrix then represented a count of the number of purchase of a size by a particular customer. Most entries were therefore 0, where they were either never bought or had been returned. The customer and product factors then allow for the recommendations to be made by taking the dot product of the customers matrix with the product matrix. The results of each method's performance for both shoes and dresses is given in the following section with discussions of our observations.

A.3 Results

The results found for the different models' performances at predicting shoe sizes is given in table A.1, and table A.2 gives the results for predicting dress sizes.

As can be seen in the tables the random forest generally performed the best, with very slightly improved accuracies when compared with the modal baseline. Matrix factorisation seemed to struggle to recommend relevant sizes and lagged behind the other two systems. For predicting shoe sizes the biggest boosts to the performance were seen when including all previous purchases (regardless of how long ago the customer bought other shoes), and when the accounts were split to allow for multiple customers and recommendations per account. While there was some improvement from rounding the half sizes, generally it wasn't as important as making sure that the initial dataset was clean and that the accounts had been correctly separated into individual customers.

For predicting the dress sizes we saw very similar results. The random forest again performed slightly better than modal baseline. The surprise was that adding keywords to the features used by the random forest (such as petite / tall etc.) actually worsened the accuracy of the predicted sizes. This suggested that adding in the extra information about the fit or type of dress was acting to increase the complexity of the model, and resulted in more branches in the random forest which split on these

Table A.1: Results of the different recommender systems at predicting shoe sizes.

	Modal Baseline		Random Forest		Matrix factorisation	
	Women	Men	Women	Men	Women	Men
Mode of all customers	0.219	0.189				
Customer's previous years purchases	0.555	0.502	0.566	0.507		
Customer's previous 3 months purchases	0.540	0.482	0.566	0.488		
Rounded sizes	0.561	0.557	0.581	0.554		
Randomly split previous purchases	0.627	0.616	0.628	0.601	0.583	0.556
Account splitting	0.755	**0.749**	**0.756**	0.740	0.661	0.691

Table A.2: Results of the different recommender systems at predicting dress sizes.

	Modal Baseline	Random Forest	Matrix factorisation
Mode of all customers	0.223		
Customer's previous 3 months purchases	0.525	0.510	0.508
Using keywords	0.525	0.498	0.508
Account splitting	0.636	**0.639**	0.554

features, but these splits weren't helpful when compared with the size features.

A.4 Summary

ASOS returns incur a significant cost for the company which leads to lost revenue. Many products are returned due to them being the wrong size, and implementing an accurate recommender system could help inform customers of the best size to buy, hence also helping to lower the number of returns.

Overall the methods tested here performed very well for an initial investigation into recommender systems for size prediction. While matrix factorisation didn't perform as well as the other methods, the random forest and simple modal baseline both did well to correctly predict the shoe size that a customer should buy, getting it right 75% of the time. Furthermore, by extending the problem to dress sizes, we showed that the methods could continue to work (albeit with a slightly reduced accuracy) for more complex products with different size conventions.

While the methods themselves worked well, the most vital stage was to ensure that the data was prepared in the most sensible way. This meant proving the models with the most useful features such as brand information and extra keywords, as well as splitting accounts which contained multiple customers to be able to give multiple recommendations depending on the customers and product.

Finally, even with a perfect recommender system, additional work would be required to build trust in the recommendations. Customers would need to be convinced that the recommendation provided would in fact be the correct size to prevent them from continuing to purchase multiple sizes, and due to personal preferences, no system would ever be able to perfectly predict what every customer would deem fit them best.

Appendix B

Forecasting ASOS Sales using Machine Learning

This appendix details the work completed at ASOS during a six-month full time industry placement undertaken at the end of the third year of the CDT's PhD program. As the placement began at the onset of the Covid-19 pandemic, the initial project was to create reliable financial forecasts which could take into account the effects of the pandemic. Initially we constructed basic forecasts to benchmark the performance of our models, before testing different types of machine learning forecasts and finally settling on a model built using the Facebook Prophet python package. Not only did the forecasts produced result in more accurate sales predictions across almost all regions where ASOS operates, but they were built with explainability as a key focus and allowed for an in-depth analysis of what contributed to the forecast which could help inform on business decisions (such as when to have promotions).

A second project was also started to investigate the related problem of predicting how much stock of each product should initially be bought. Using attributes of every item sold by ASOS, we created models to predict the number of units of each product which would be sold and therefore how much stock should be purchased. While working on this project the baselines and first models created showed great promise and acted as the proof of concept needed to display how useful the models could be. Although the placement ended before a final dashboard could be created which would have been the end business product used by ASOS buyers when making the stock purchases, the groundwork completed left the project in a good state to be continued to produce a helpful stock prediction tool.

B.1 An introduction to forecasting

Forecasting is a form of time-series analysis whereby past data is used to make predictions about the future. As such it is often central to many business activities where planning about the future is paramount to be able to effectively allocate resources, set goals, and measure performance. At ASOS, forecasting the number of sales made in each region is an incredibly important task, and is not only required for the planning of many aspects of the business, but it also allows us to determine the company's growth.

At the beginning of 2020, the Covid-19 pandemic was beginning to impact all facets of life. In the United Kingdom the first 'lockdown' began in March (just after the start of the placement) and the shopping habits of millions was forced to change. On top of this, social distancing measures limited the number of workers who were able to work in ASOS's fulfillment centres. With so much uncertainty and events rapidly changing, there was high demand for forecasts which were able to take into account some of the effects of Covid-19. These forecasts could help ASOS stay ontop of the situation, knowing how many sales they were likely to make which would give an indication to the financial situation as well as provide information about how large the backlog of orders could become and hence how delayed the fulfillment could be.

There are many possible way to create forecasts. At ASOS, the finance team works to create forecasts using their domain knowledge. Roughly speaking they can examine the previous weeks data and by knowing about promotional events and holidays, as well as comparing with the same period from the previous year, a forecast can be hand made to predict the number of sales they expect in the coming week(s). The aim of this project was to replicate the finance team's forecast with an automated machine learning model that could also take into account new features, such as the effects of Covid-19. In the next section I describe the methods we used to create forecasts, before presenting the results of the forecasts in section B.3, and summarising the project in section B.4.

B.2 Forecasting Methods

Before attempting to create an intelligent machine learning model to generate the forecasts, it was first necessary to gather the data which was used by the models, and understand how we could measure their performance.

B.2.1 Data

The data was predominantly made from the historical sales data of ASOS. Every sale is recorded with details about time and region where the purchase was made, and this data is collated into daily sales figures for each region. These daily inputs were then used as the historic data which the models learnt from.

While a forecast could be made with only the daily sales data, we wanted to capture additional information about what else affected sales. As the focus was during the start of the Covid-19 lockdown, the first additional dataset included was the daily Covid-19 case numbers taken from Dong et al. (2020). Although this gave information about the number of people getting sick, it didn't explicitly inform about the lockdown which is what had the greatest overall effect (with people not leaving their homes and physical clothes stores being closed). As such, an additional 'day zero' was defined which was taken to be when lockdown began. By including this information as a one-hot encoded feature (with a 0 before day zero and 1 after) the models could then attribute the effect on the sales from that day specifically to the lockdown.

One of the most important features which impacted sales at ASOS was the promotions calendar. This detailed the sales promotions which regularly ran and offered customers discounts such as $x\%$ off everything, or free delivery on spends over £X. As expected, on days with large discounts there would be an increase in sales, and these appeared as spikes in the time-series. Similar to the day zero feature, the promotions were one-hot encoded to show if there was a promotion, and if so, what kind of discount was being offered. Other features were tested later in the investigation, including the weather which was assumed would have the greatest impact out of these additional features. To include weather data, we obtained daily historic temperatures, precipitation, humidity, wind speeds, and cloud cover for each of the regions where we were producing sales forecasts.

However, simply including the historic data of the features wasn't enough to produce a forecast for the sales. To predict the sales in the future, the future values of the included features were also needed. This meant that we needed forecasts for all features before we could begin forecasting the sales. For the features such as the promotions, this was trivial (in theory) as the calendar when promotions would be applied was controlled by ASOS and as such the calendar for the future months was available to input. Unfortunately ASOS doesn't have the same control over the other features such as weather and covid cases, and instead forecasts of these features were used.

The weather forecasts could be obtained from the same source that provided the historic data, and allowed for reasonably accurate predictions to be made (at least for the short term forecasts of a week), and for long range forecasts of months in the future average temperatures etc. could be used from previous years. Forecasting covid cases was something which many research groups worked on, but to save time for the purpose of the sales forecast we assumed that covid cases remained roughly constant in the short term forecasts. This allowed for the forecasts to use the full range of features which we also estimated for the future.

B.2.2 Metrics

To measure the performance of the forecasts produced we calculated various metrics which compared the true sales to the sales predicted by the forecast. By also comparing with the performance of the finance department's forecast, we had a good indication of how our models performed. As a regression problem (where a continuous value was being predicted from the features) we could have used any error metric such as the mean absolute error (MAE), or mean squared error (MSE). However, the most commonly used metric for forecasts is the mean absolute percentage error (MAPE) which is defined as:

$$MAPE = \frac{100}{N} \sum_{i=1}^{N} |\frac{y_i - \hat{y}_i}{y_i}|, \tag{B.1}$$

where y_i is the true value and $\hat{y}_i$ is the forecast value.

To stay consistent with the finance department and give an easily understandable single value, the MAPE was used as a quick indication of how the forecasts performed.

We tested several different types of forecasts, starting with a very basic model which simply took the previous weeks sales to use as a baseline, we then moved onto using machine learning algorithms to attempt to include more features. A random forest was implemented which managed to perform slightly better than the simple baseline, however, the method settled on was built using Facebook Prophet (Taylor & Letham, 2018).

B.2.3 Facebook Prophet

Facebook Prophet (FBProphet) is a forecasting package developed by data scientists from Facebook who aimed to create a flexible forecasting package capable of providing useful forecasts for a range of time-series datasets. It was designed to use intuitive parameters which could be adjusted by scientists with domain knowledge without having to create a new underlying model.

The model is a structural time-series model (Harvey & Peters, 1990) where the model can be decomposed into three main components: the trend, seasonality, and holidays, which are combined as follows.

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t. \tag{B.2}$$

In this equation the trend, $g(t)$, models the non-periodic changes in the data; the seasonality, $s(t)$, models the periodic changes which can be daily / weekly / yearly or any other set time period; the holidays, h(t), represent the effect of special holidays which can occur on irregular time frames over a set number of days. Finally, ε_t gives an error term for any additional changes not accounted for in the model (and is assumed to be normally distributed).

The trend can be modelled in two ways using FBProphet, it can either be a saturating growth model, or a linear model. For growth forecasting the trend is typically modelled using a logistic model which has the form:

$$g(t) = \frac{C}{1+\exp\left(-k(t-m)\right)}, \tag{B.3}$$

where C is the carrying capacity, k is the growth rate, and m is an offset parameter.

However, there are two important differences made in this model. First, the carry capacity is not constant (as the total number of people who use ASOS is not constant) and thus the constant term C is replaced with the time dependant capacity $C(t)$. Second, the growth rate is not a constant as a company's growth will change depending on many factors and the growth rate needs to be adaptable to fit to the historic data. To incorporate the changes to the trend, FBProphet uses explicitly defined 'changepoints'. These changepoints are points of time where the growth rate is allowed to change.

For S changepoints at times s_j, a vector of adjustment rates, δ, is defined where each δ_j represents the change in growth rate at time s_j. This allows the new growth rate to be written as the base rate, k, plus any adjustment made up to that time: $k+\sum_j \delta_j$. We can also define the vector, $\mathbf{a}(t)$ as

$$a_j(t) = \begin{cases} 1, & \text{if } t \geq s_j \\ 0, & \text{otherwise} \end{cases}, \tag{B.4}$$

which allows the rate to be written as: $k+\mathbf{a}(t)\delta$. The offset parameter, m, must also be adjusted to then connect the endpoints and can be calculated as follows

$$\gamma_j = (s_j - m - \sum_{l<j} \gamma_l)(1 - \frac{k+\sum_{l<j}\delta_l}{k+\sum_{l\leq j}\delta_l}), \tag{B.5}$$

and the trend model can then be written as:

$$g(t) = \frac{C(t)}{1+\exp\left(-(k+\mathbf{a}(t)\delta)(t-(m+\mathbf{a}(t)\gamma))\right)}. \tag{B.6}$$

Instead for the linear trend model we use:

$$g(t) = (k + \mathbf{a}(t)\delta)t + (m + \mathbf{a}(t)\gamma), \tag{B.7}$$

where the only difference in terms is that the adjustment γ_j is instead set to $-s_j\delta_j$.

The seasonality is modelled using the following equation:

$$s(t) = \sum_{n=1}^{N} (a_n cos(\frac{2\pi nt}{P}) + b_n sin(\frac{2\pi nt}{P}), \tag{B.8}$$

where P is the period of the seasonality (e.g. $P = 365.25$ for a yearly seasonality, 7 for weekly etc.), and the parameters $\boldsymbol{\beta} = [a_1, b_1, ..., a_N, b_N]$ are estimated by taking $\boldsymbol{\beta}$ to be normally distributed. For a given N, eg N = 3, we can then write the seasonality in terms of matrix of seasonality vectors, $X(t)$ where

$$X(t) = [cos(\frac{2\pi(1)t}{P}), ..., sin(\frac{2\pi(3)t}{P}], \tag{B.9}$$

thus making the seasonality

$$s(t) = X(t)\boldsymbol{\beta}. \tag{B.10}$$

Finally, the holidays can be modelled in a similar vein, where for each holiday, i, we define D_i to be the set of dates (past and future) for that holiday. An indicator function is used which represents whether the time, t, falls on a holiday where

$$Z(t) = [\mathbf{1}(t \in D_1), ..., \mathbf{1}(t \in D_N)], \tag{B.11}$$

and each holiday is assigned a parameter κ_i which represents the corresponding change to the forecast and is also taken to be normally distributed. This allows the holidays effects to be written as

$$h(t) = Z(t)\boldsymbol{\kappa}. \tag{B.12}$$

B.3 Results

To prevent revealing any potentially sensitive information, only a brief description of the results found during the forecasting project is given with any sales figures omitted.

The FBProphet models were very successful at producing forecasts, often outperforming the hand made finance team's forecasts in terms of MAPE. Figure B.1 shows an example of a typical set of weekly forecasts compared with a longer term monthly forecast. It also shows the finance team's weekly forecasts during the same periods and true value for the sales. It is clear from this figure that the FBProphet models were working well, outperforming the finance team in weeks 1&4 and not being too far off during weeks 2&3. Indeed, the shape of the weekly FBProphet model was often more similar to the true sales, however, its sensitivity to the initial sales meant that it was prone to then consistently over (or under) predicting the sales.

The results from this random month period are also given in table B.1 which combined the MAPEs of the weekly forecasts to give a single MAPE for the month long period. The table highlights how similar the forecasts were, and shows that the FBProphet models were performing well, with both the weekly models and monthly model able to give good results. While taking the combined MAPEs over the month is slightly unfair for the weekly models (as they performed much better for a couple of the weeks and had the overall MAPE lowered by the worse performance in the other months), it allowed them to be compared with the longer FBProphet model.

While the FBProphet model was able to match the finance team's forecasts, the major benefit of using FBProphet was the explainability of the model's predictions. Using FBProphet it was possible to obtain the feature importances for each aspect of the model and hence determine what the model believed would affect the sales. This also allowed waterfall plot such as the one displayed in figure B.2 to be made which gave a visual breakdown of how the model came to its prediction of sales for a given day.

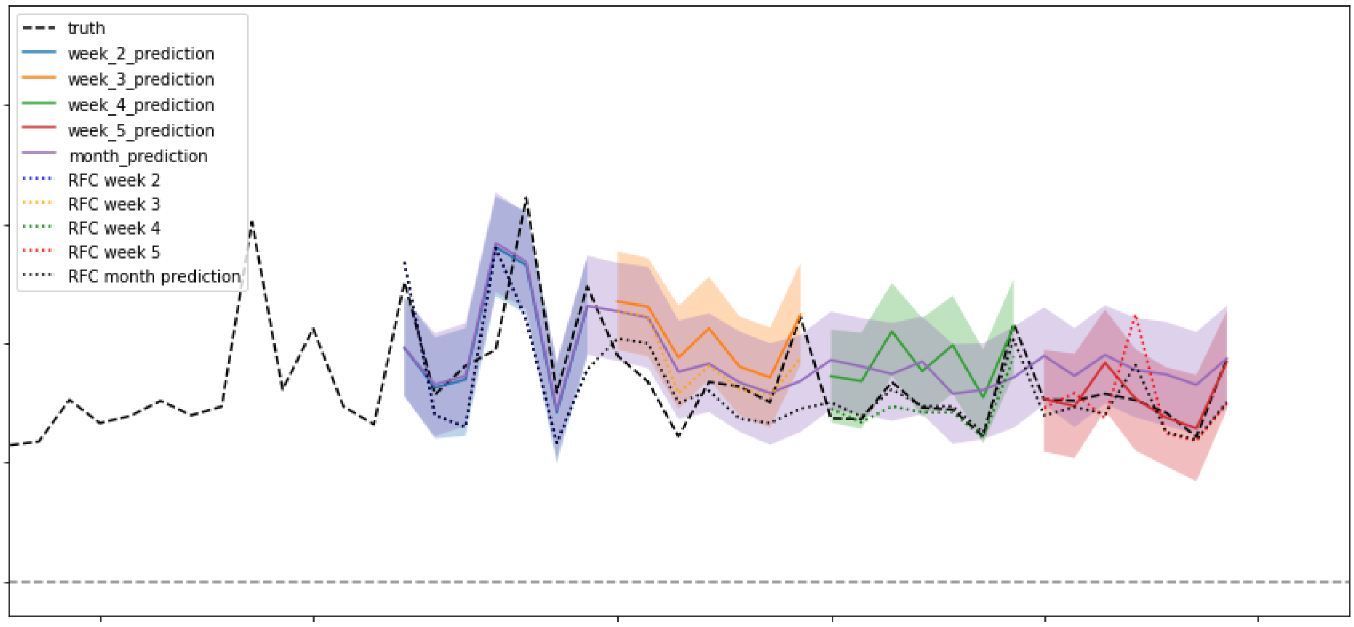

Figure B.1: Figure showing a random set of forecasts comparing a weekly FBProphet model to a monthly model, along with the weekly finance department's forecasts and the true sales.

Table B.1: Results of the MAPE achieved by the different forecasts generated over a random month.

	FBProphet 4x1 weeks	**FBProphet month**	**finance 4x1 weeks**
APAC	0.316531	0.423905	**0.282204**
Australia	0.168922	**0.150393**	0.235513
Benelux	0.228464	0.231321	**0.208171**
France	0.419565	0.331926	**0.254686**
Germany	0.185199	0.211124	**0.145718**
Ireland	**0.225047**	0.23762	
Italy	**0.213963**	0.301651	0.236378
MENA	0.204605	**0.158024**	0.331063
ROE	0.177394	0.170619	**0.164534**
ROW	0.255996	**0.158361**	0.511075
Russia	0.461566	0.299544	**0.254753**
Scandinavia	**0.17258**	0.185033	0.207391
Spain	0.198549	**0.184263**	0.316124
United Kingdom	0.168453	0.193857	**0.153485**
United States	0.205302	0.233086	**0.167667**

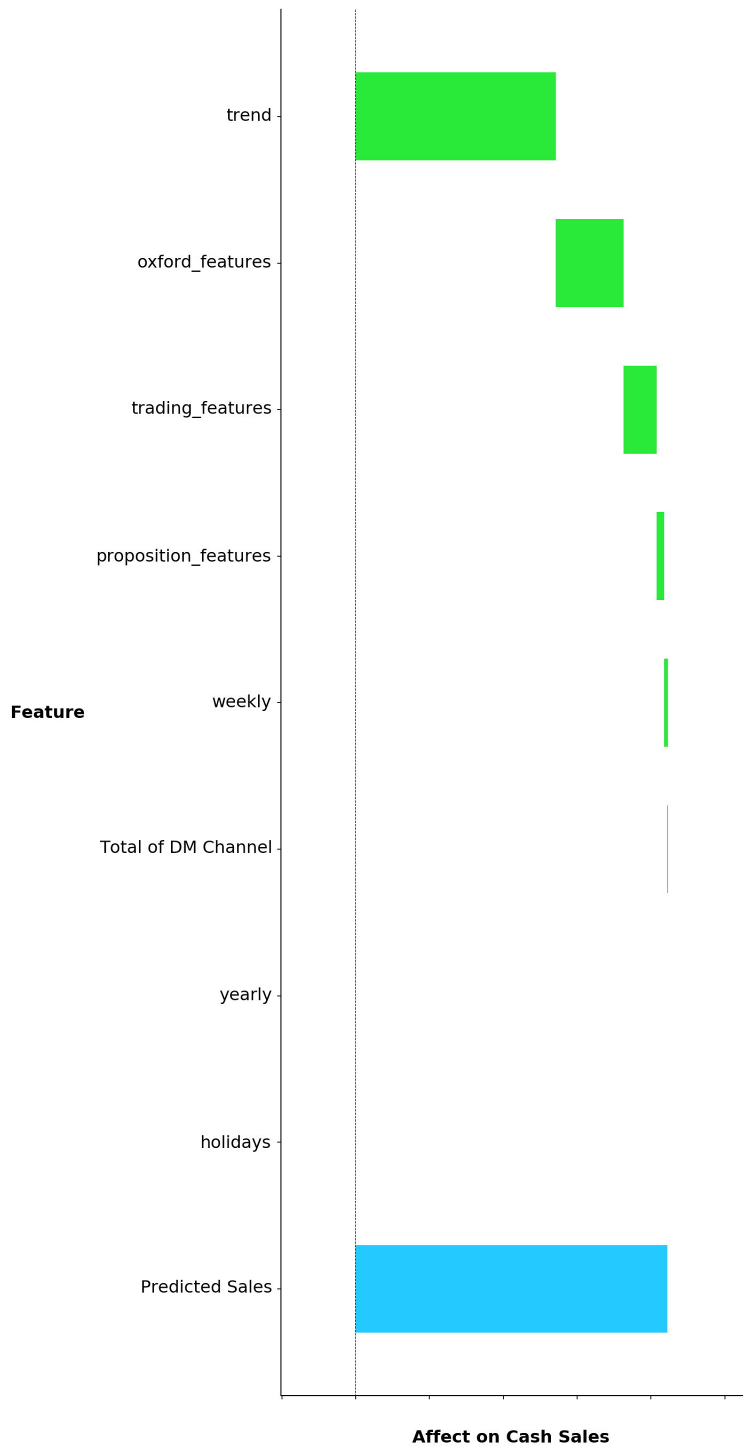

Figure B.2: An example waterfall plot for a random day which shows how the FBProphet model weighted the features and came to its prediction.

B.4 Summary

The main goal of the project to create explainable forecasts which could take into account a wide range of features was definitely achieved. The FBProphet models were able to use the Covid-19 case data to determine the impact on sales, and by also including other features such as the trading calendar, which detailed the promotions, as well as the delivery proposition and later the weather, the model was able to provide useful insights into the impacts on sales.

The FBProphet forecasts created were able to perform well and when compared with the finance department's forecasts they were able to match their performance. The added flexibility of being able to quickly add extra features as well as being able to show the feature importance made the model very capable of being used to help inform on a variety of aspects of ASOS's business.

B.5 An introduction to stock prediction

Predicting the amount of stock to initially purchase for each product is another crucial area of ASOS's business. If too much stock is purchased the excess products can end up costing the company more than they make back in sales, and on top of the cost, they take up room in warehouses and can end up having a greater environmental impact. If too little stock is purchased then there is a missed opportunity for additional sales of the product and a greater profit would have been possible. The consequences and impacts of both scenarios make it such an important area that only highly trained expert buyers are able to make the decisions about purchasing the initial stock.

These buyers use their domain knowledge of how previously listed, similar products have sold, and use these similar products to predict how much stock they think the new products could sell and therefore how much initial stock to buy. This is a process which could be replicated using machine learning. The model would first find the most similar products to the new product of interest, and could then use the sales of those similar products to make a prediction about how much the new product will sell. As such we set about creating a model which could then be used by the expert buyers to help inform them on how much stock to buy.

B.6 Methodology

The problem was split into two parts, an information retrieval task whereby the most similar products for a given product of interest would be returned, and the prediction stage where a machine learning algorithm would predict the stock to be purchased.

To begin, a baseline model was made which used basic filters to find the most similar products and simply predicted the initial stock to be the same as for the most similar product. The filters worked by taking the product attributes and finding the items where there was the greatest overlap of attributes. Initially the attributes used were simply the product's division, category, brand, colour, range, price (both cost price and sale price), and date, and as seen in the results these were able to provide quite sensible suggestions for the most similar products.

An extension which was being tested was using a more advanced system of obtaining product attributes directly from images of the items. This could allow for a deep learning algorithm to provide more details about the products and find more relevant items, or even directly give the most visually similar products.

With the most similar product(s) identified, a model could then be used to predict the initial stock to purchase. For our baseline we simply took the same initial stock as the most similar item, however, this would obviously not be the best prediction for many different products and instead it was hoped that a machine learning algorithm could improve upon the baseline. By providing the initial stock, sales, and attributes of the most similar products as features to the algorithm we then tried to predict the sales in the first month of the product of interest being sold which could then be taken as the recommendation for the initial stock.

B.7 Results

Similar to the forecasting project, to not give away any potentially sensitive information I present only a brief summary of the results found during the investigation of stock prediction.

	baseline (average sales of filtered products)	Boosted decision tree
MAE	326.30	131.85

Table B.2: Results of the first baseline (using the average sales of the most similar filtered products) and the boosted decision tree when predicting the number of sales of dresses.

As an example of the filters used to obtain similar products, figure B.3 shows a sample of products and the retrieved most similar product. In general the filtering process worked very well, and as seen for a typical pair of trainers the most similar item returned was a variation of the trainers from the same brand. The filters worked similarly for the hair care product, with the most similar product returned being a smaller bottle of the same product. However, for the belt, the most similar product by the filters was a set of face masks. This highlighted a key failing of the filters which treated everything in the category of accessories as being similar, and so to get better results for the similar products further sub categories would be required.

Using dresses as an example product dataset we then used the filters to obtain the most similar dresses for each product. To give a baseline error we then took the average sales of the most similar dresses to predict the number of sales of the dress of interest and calculated the mean absolute error (MAE), and the results are given in table B.2.

This allowed us to compare a simple boosted decision tree which was used as the first machine learning model tested. The boosted decision tree took the various attributes of the dresses (the division, sub-category, colour, brand, range, dress type, fabric type, length, sleeve length, body type, sale price and cost price) as the features, and use these to create a mapping to the number of units sold. As seen in table B.2 the boosted decision tree was able to massively improve the prediction of the number of dresses sold, having a MAE 2.5$\times$ lower. Although the MAE remained fairly high at over 100 units, this isn't too bad a result as many dresses in the dataset had thousands of sales.

An additional benefit of using the boosted decision tree was the ability to easily examine the feature importances. In our investigation we found that the most useful attributes were the prices and the brand. This suggested that the most useful

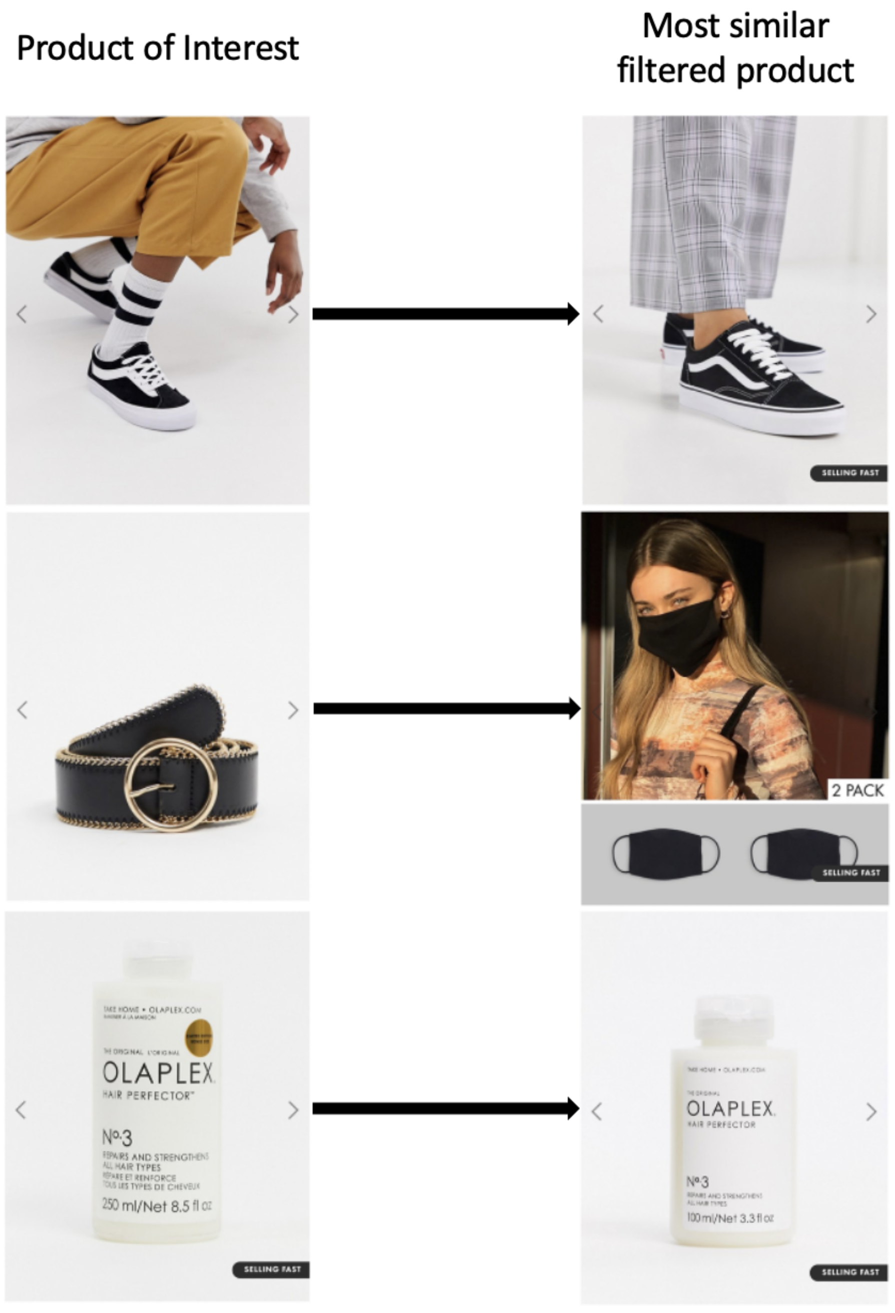

Figure B.3: Figure showing examples of the filtering process to obtain similar products.

information which would inform about the number of sales that could be expected was the cost and sale prices as well as the brand of the dress.

B.8 Summary

While the project was still in its early days, the initial results from the baseline and boosted decision tree were very encouraging. The fact that simple filters were able to extract similar products from their attributes, and that the machine learning model was able to produce a reasonable estimate for the number of sales from the same attributes meant that already the proof of concept had been achieved.

The next stages were to add other features to the model, and use the information about the similar products in the machine learning model. By adding in this extra information (such as the number of units the similar items sold) we could expect another decent improvement in the error. By continuing with the work and improving the models used, this would lead to a very helpful tool which could be used by the expert buyers to make more quantifiable decisions about the initial stock to buy.

Appendix C

Correlations between UV-index and Covid-19 Cases in the UK

This appendix is based on work completed in collaboration with A. Blum, C. Nicolaou, and O. Lahav, in which we investigated the correlation between UV index and Covid-19 cases in the United Kingdom (UK) (Blum et al., 2020). During this collaboration I worked predominantly on the statistical analysis of the Covid-19 cases and UV index. It was well established early on in the pandemic that the rate of Covid-19 infections could be suppressed with social distancing measures, however, there wasn't anywhere near as much investigation into what environmental effects could impact the spread of Covid-19. Here I discuss our study into the correlation between natural ultra-violet (UV) radiation from the Sun and Covid-19 cases in the UK. By examining the daily case rates of Covid-19 infections (F) and the UV-index (UVI) between January 2020 to March 2021, we found a strong anti-correlation between log_{10}(F) and log_{10}(UVI) of -0.934 in the period from the end of the first lockdown in the UK on the 11^{th} of May 2020 to the 10^{th} of March 2021. This anti-correlation could reflect a causation (along with other factors correlated to UVI) with the reduction seen in Covid-19 infections and highlight the need to include as many additional factors as possible when modelling Covid-19 infections and deaths.

C.1 Introduction

Our hypothesis is that natural UV light suppresses the spread of Covid-19 virus in at least two ways: the effect on the virus itself, and on the human skin. We note that while natural UV has the potential to cause skin cancer, it also generates vitamin-D which supports the immune system. There are three types of solar UV radiation classified according to their wavelength. UVC is a short wavelength (100 - 280 nm) radiation and it is the most damaging to the human body. However, it is completely filtered by the atmosphere and does not reach the Earth's surface, so while it is well known that UVC produced in the lab is used to inhibit viruses, it cannot have the same effect in nature. UVB is a medium wavelength (280 - 315 nm) radiation, and similar to UVC, most of it is filtered by the atmosphere. UVA is a long wavelength (315 - 400 nm) radiation and accounts for about 95% of the UV radiation reaching the Earth's surface. The UV Index (UVI) is a measure of the strength of sunburn-producing UV radiation at a particular place and time, with typical values of UVI in the UK ranging between 0 and 8.

The possible correlation between UV light and Covid-19 has been discussed in the literature, with contradictory conclusions. One study by Yao et al. (2020) found no association of Covid-19 transmission with UV radiation in Chinese cities. Others found modest impact of UV light and other environmental effects on the reduction of Covid-19 transmission (Xu et al., 2020). Another study by Yudistira et al. (2020) pointed out that UV radiation will not be effective in places with high air pollution, where UV light turns into heat. On the other hand, other studies have found that UV light is associated with decreased Covid-19 growth rate. Given the disagreement on the impact of UV light on Covid-19 transmission we take a fresh look at data for the UK. Our study was also strongly motivated by the second wave of Covid-19 in many Northern hemisphere countries during the winter.

Most viral respiratory infections have a seasonal pattern that may be related to climate changes, humidity, UV irradiation from the sun, latitude, air pollution, height, and the human nature (genetic and epigenetic factors and behavioural characteristics). Enveloped viruses have a cold temperature preference (such as influenza A and B). A study that examined the climate of 50 cities affected by Covid-19 found that 8 cities had particularly high mortality rates. All 8 of these cities were located between latitudes 30°N and 50°N, with a temperature between 5°C to 11°C, and low humidity. Countries located below latitude 35°N had lower Covid-19 mortality rates. This could be due to the fact that countries located above 35°N have insufficient sunlight necessary for vitamin D activation.

Vitamin D deficiency was found to correlate with hypertension, diabetes mellitus, obesity, and is associated with increased mortality rates. Countries that suffered the highest mortality are known to have a high prevalence of vitamin D deficiency (Italy, Spain, UK, France). In Nordic countries, where sunlight is limited, vitamin D food fortification is mandatory, and the mortality rate was lower in the recent pandemic. Looking at specific Italian cities, Milan's latitude is 45°N, and Naples is located at 40°N. This means that Naples gets 58 more sunny days annually compared with Milan, and in Naples the death toll from Covid-19 was 403/million compared with 15,729/million in Milan (a $>$ 39 fold increase).

Sunlight activates Nitric Oxide (NO) in the skin. NO is a potent modulator of the cardiovascular system, reducing blood pressure and peripheral resistance. NO is a signalling molecule responsible for the hoemostasis of blood vessels. It affects cellular proliferation, inflammatory processes, and has an anti-bacterial and anti-viral activity. Usually NO is produced in endothelial cells by oxidation of L-arginine to NO and citrulline. Inducible NOS (i-NOS) is calcium-independent, and is activated during stress conditions like acute or chronic inflammation or infection. NO helped prevent the replication of SARS-CoV by inhibiting fusion between the S protein and its receptor and stopping the viral RNA replication. UVA penetrates into the dermal layer through the keratinocytes to the fibroblasts and themicrovascular endothelial cells. Keratinocytes, Langerhans cells, dermal fibroblasts, and melanocytes are all cells that have the ability to induce i-NOS once activated by cytokines.

In our study we focus on highlighting the link between UV and Covid-19 infections in the UK, using Covid-19 data from Dong et al. (2020), and UVI data for

London from Van Geffen et al. (2017). Although the UVI used was for London, the variations in different places in the UK were within 0.8 UVI. The stringency index represented the lockdown measures taken in the UK and were defined in Hale et al. (2021).

C.2 Results

Figure C.1 shows the daily infections and deaths for the UK along with the UVI and stringency index for the period 22nd January 2020 to 10th March 2021. As expected from January to July 2020 the UVI increased to a peak before falling. We can see that during the summer when the UVI was high the infections were at a minimum. However, in the initial increase in UVI the number of daily infections was seen to also rise until April. The UK introduced the first lockdown on the 23rd March which is what resulted in a decrease in the number of infections due to social distancing. While the UVI might have also helped lower the number of infections, it was clear that the main factor in controlling the spread of Covid-19 was social distancing.

The lockdown was relaxed on 11th May, and while cases remained low for much of the summer, over the period 2nd July to October the increase in infections was strongly anti-correlated with UVI. In November there was a decrease in the number of cases due to another national lockdown being imposed, but then the cases increased again once the lockdown was lifted. During this period the UVI was also low and therefore wouldn't have had an impact in reducing cases.

To quantify the correlation seen in Figure C.1, we consider the correlation coefficient between X and Y, defined as

$$\rho = \frac{\langle (X-\mu_X)(Y-\mu_Y)\rangle}{\sigma_X \sigma_Y}, \tag{C.1}$$

where μ represents the mean and σ the standard deviation. We applied this to $X = log_{10}$(UVI) and $Y = log_{10}$(F), and found the correlation coefficient averaged over the ranges given in table C.1. We also investigated the correlation coefficient for different time lags of 7 and 14 days (in case the effect of UV was more obvious after a set amount of time). However, we saw that the time lag made very little

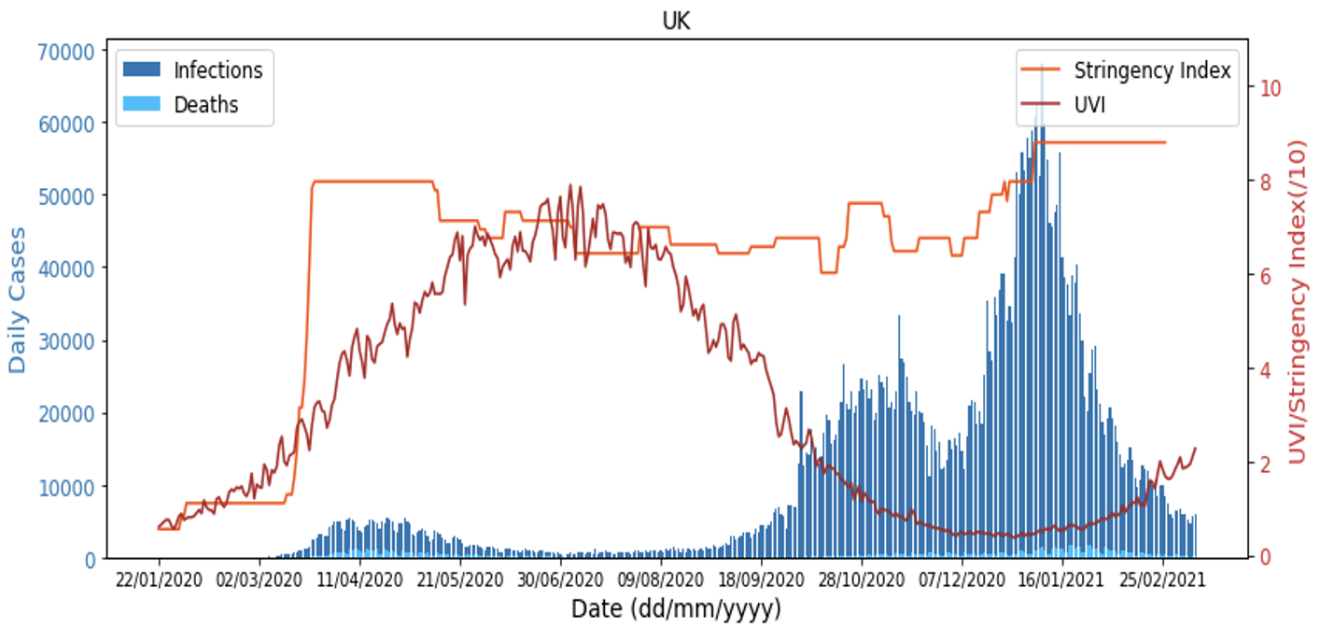

Figure C.1: Figure showing the daily number of Covid-19 infections and deaths along with the UV index and stringency index for the UK in the period 22nd January 2020 to 10th March 2021.

Date Range	Corr. Coefficient of log_{10}(UVI) & log_{10}(F)		
	No lag	7 day lag	14 day lag
23 Mar 20 - 10 Mar 21	-0.917	-0.910	-0.886
11 May 20 - 10 Mar 21	-0.934	-0.922	-0.896
02 Jul 20 - 10 Mar 21	-0.927	-0.910	-0.879
02 Jul 20 - 28 Oct 20	-0.964	-0.958	-0.943
28 Oct 20 - 10 Mar 21	-0.751	-0.738	-0.611

Table C.1: Table of correlation coefficients for different date ranges and with different time lags between Covid-19 cases and UVI

difference to the correlation coefficients, probably as competing effects washed out any particular time lag.

We also plotted the daily infections and UVI with the rolling correlation coefficient between log_{10}(UVI) and log_{10}(F) with a window size of 50 days in figure C.2. This plot highlights the overall negative correlation between the infections and UVI from mid-April onwards. However, it also clearly displays that in November and December, there was a positive correlation due to the fact that lockdowns were imposed which led to a reduction in number of cases.

Finally, we plotted the log-log scatter of F against UVI in figure C.3, display-

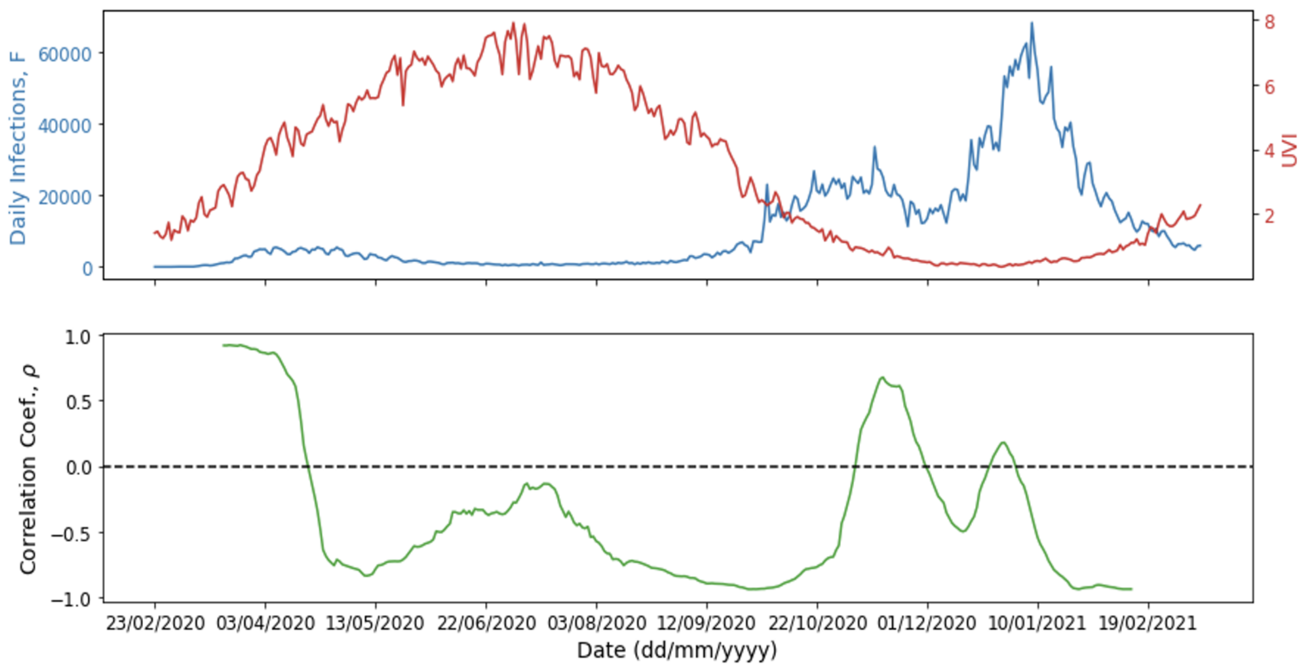

Figure C.2: In this figure the top panel shows the daily infections (blue) and UVI (red) for the UK from the 23rd February until the 10th March 2021, and the bottom panel gives the rolling correlation coefficient of log_{10}(F) and log_{10}(UVI) with a window size of 50 days.

ing the time intervals with different colour codings. The near linear correlation is remarkable, and we fit the data to

$$log_{10}(F) = m log_{10}(UVI) + c, \tag{C.2}$$

using standard least squares regression. For the period between the 23rd March 2020 to the 10th March 2021 we found $m = -1.292 \pm 0.029$ and $c = 4.199 \pm 0.017$ with the 68% confidence level derived using bootstrapping. The three fits given in figure C.3 were within 10% of the slope and 1% of the intercept.

C.3 Summary

In this study of finding correlations between the UVI and Covid-19 infections we saw a significant negative correlation of -0.934 between the 11th May 2020 and 10th March 2021. It is important to note the well known fact that a correlation between two observables does not necessarily mean causation. We emphasise that the

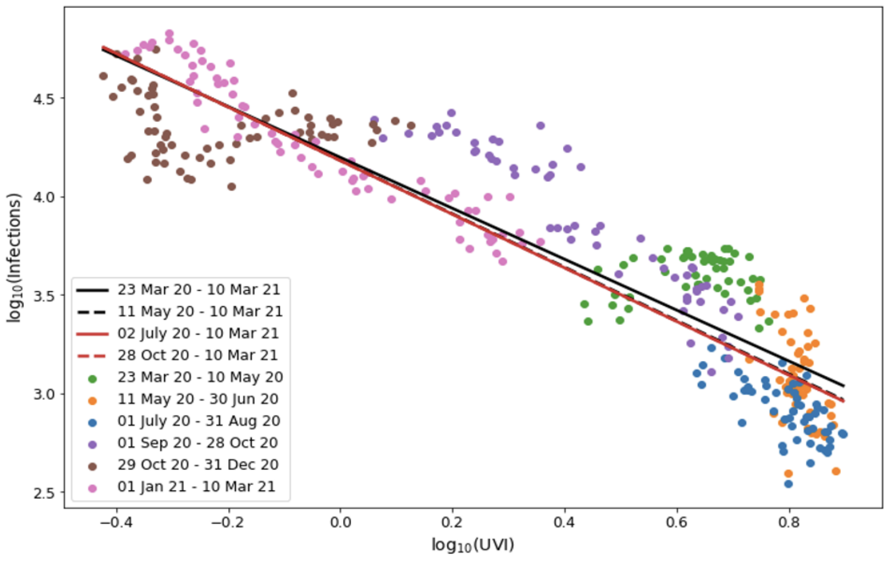

Figure C.3: Figure showing log-log scatter plots of F against UVI with colour-coded time intervals, where the lines give the fits by linear regression for different time periods.

Covid-19 infection rate is a multi-parameter problem, and there is strong empirical evidence that social distancing, wearing masks, and vaccination programs are the most significant factors in reducing transmission.

However, the strength of the correlation observed in our study cannot be ignored. When the UVI increased above a certain level, there was a decrease in the rate of infection, and when the UVI decreased below a certain level, the rate of infection increased. While the lockdown was the most important factor in reducing the infection rate, the much lower rate during the summer months (when the lockdown had ended) may suggest that environmental factors had an effect on the infections, and UVI in particular was a dominant factor in those summer months.

The negative correlation detected between UVI and infection rate could also suggest that the rate of change of UVI may trigger the viral infectivity mechanism. If the UV light does directly affect the level of infections it would be by either reducing the survival of the virus itself, or by improving the immunity of subjects (via the production of Vitamin D and activation of the NO pathway), or a combination of the two. Future work would include adding other environmental variables that may have an effect on the rate of infections such as humidity, and air pollution.

Bibliography

Abadi M., et al., 2015, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, `https://www.tensorflow.org/`

Abbott B. P., et al., 2016, Phys. Rev. Lett., 116, 061102

Abbott B. P., et al., 2017, Nature, 551, 85

Abbott T., et al., 2018, The Astrophysical Journal Supplement Series, 239, 18

Abdalla F. B., Banerji M., Lahav O., Rashkov V., 2011, Monthly Notices of the Royal Astronomical Society, 417, 1891

Adams J. C., 1846, Monthly Notices of the Royal Astronomical Society, 7, 149

Ahumada R., et al., 2020, The Astrophysical Journal Supplement Series, 249, 3

Aihara H., et al., 2018, Publications of the Astronomical Society of Japan, 70, S4

Alam S., et al., 2015, The Astrophysical Journal Supplement Series, 219, 12

Almosallam I. A., Jarvis M. J., Roberts S. J., 2016, Monthly Notices of the Royal Astronomical Society, 462, 726

Altman N. S., 1992, The American Statistician, 46, 175

Amendola L., et al., 2018, Living reviews in relativity, 21, 2

Arnouts S., Cristiani S., Moscardini L., Matarrese S., Lucchin F., Fontana A., Giallongo E., 1999, Monthly Notices of the Royal Astronomical Society, 310, 540

Bailey E., Batygin K., Brown M. E., 2016, The Astronomical Journal, 152, 126

Ball N. M., Brunner R. J., Myers A. D., Strand N. E., Alberts S. L., Tcheng D., Llorà X., 2007, The Astrophysical Journal, 663, 774

Bannister M. T., et al., 2016, The Astronomical Journal, 152, 70

Bannister M. T., et al., 2018, The Astrophysical Journal Supplement Series, 236, 18

Bartelmann M., Schneider P., 2001, Physics Reports, 340, 291

Batygin K., Brown M. E., 2016a, The Astronomical Journal, 151, 22

Batygin K., Brown M. E., 2016b, The Astrophysical Journal Letters, 833, L3

Batygin K., Morbidelli A., 2017, The Astronomical Journal, 154, 229

Batygin K., Adams F. C., Brown M. E., Becker J. C., 2019, Physics Reports

Beck R., Dobos L., Budavári T., Szalay A. S., Csabai I., 2016, Monthly Notices of the Royal Astronomical Society, 460, 1371

Becker J., et al., 2018, The Astronomical Journal, 156, 81

Benitez N., 2000, The Astrophysical Journal, 536, 571

Bernardinelli P. H., et al., 2020a, The Planetary Science Journal, 1, 28

Bernardinelli P. H., et al., 2020b, The Astrophysical Journal Supplement Series, 247, 32

Bernstein G., Khushalani B., 2000, The Astronomical Journal, 120, 3323

Betoule M., et al., 2014, Astronomy & Astrophysics, 568, A22

Bettaney E. M., Hardwick S. R., Zisimopoulos O., Chamberlain B. P., 2019, arXiv preprint arXiv:1904.00741

Beust H., 2016, Astronomy & Astrophysics, 590, L2

Blum A., Nicolaou C., Henghes B., Lahav O., 2020, medRxiv preprint medRxiv:2020.11.28.20240242

Bojarski M., et al., 2016, arXiv preprint arXiv:1604.07316

Bolzonella M., Miralles J.-M., Pelló R., 2000, Astronomy & Astrophysics, 363, 476

Brammer G. B., van Dokkum P. G., Coppi P., 2008, The Astrophysical Journal, 686, 1503

Breiman L., 1996, Machine Learning, 24, 123

Breiman L., Spector P., 1992, International statistical review/revue internationale de Statistique, pp 291–319

Breiman L., Friedman J., Stone C. J., Olshen R. A., 1984, Classification and Regression Trees. Routledge

Brown M. E., 2017, The Astronomical Journal, 154, 65

Brown M. E., Batygin K., 2019, The Astronomical Journal, 157, 62

Bruzual A., et al., 1983, The Astrophysical Journal, 273, 105

Burkov A., 2019, The hundred-page machine learning book

Cáceres J., Gomes R., 2018, The Astronomical Journal, 156, 157

Carrasco Kind M., Brunner R. J., 2013, Monthly Notices of the Royal Astronomical Society, 432, 1483

Cavuoti S., Amaro V., Brescia M., Vellucci C., Tortora C., Longo G., 2016, Monthly Notices of the Royal Astronomical Society, 465, 1959

Cavuoti S., Brescia M., Amaro V., Vellucci C., Longo G., Tortora C., 2017, arXiv e-prints arXiv:1706.03501

Chandola V., Banerjee A., Kumar V., 2009, ACM computing surveys (CSUR), 41, 1

Chollet F., et al., 2015, Keras, `https://keras.io`

Christy J. W., Harrington R. S., 1978, The Astronomical Journal, 83, 1005

Cole S., et al., 2005, Monthly Notices of the Royal Astronomical Society, 362, 505

Colless M., et al., 2001, Monthly Notices of the Royal Astronomical Society, 328, 1039

Collister A. A., Lahav O., 2004, Publications of the Astronomical Society of the Pacific, 116, 345

DES Collaboration 2005, International Journal of Modern Physics A, 20, 3121

DES Collaboration 2016, Monthly Notices of the Royal Astronomical Society, 460, 1270–1299

DES Collaboration 2021, arXiv preprint arXiv:2105.13549

Dawson K. S., et al., 2012, The Astronomical Journal, 145, 10

Dawson K. S., et al., 2016, The Astronomical Journal, 151, 44

De Jong J. T., et al., 2015, Astronomy & Astrophysics, 582, A62

Dewdney P. E., Hall P. J., Schilizzi R. T., Lazio T. J. L., 2009, Proceedings of the IEEE, 97, 1482

Dicke R. H., Peebles P. J. E., Roll P. G., Wilkinson D. T., 1965, The Astrophysical Journal, 142, 414

Dong E., Du H., Gardner L., 2020, The Lancet infectious diseases, 20, 533

Dongarra J. J., Moler C. B., Bunch J. R., Stewart G. W., 1979, LINPACK users' guide. SIAM

Dongarra J. J., Luszczek P., Petitet A., 2003, Concurrency and Computation: practice and experience, 15, 803

Drlica-Wagner A., et al., 2018, The Astrophysical Journal Supplement Series, 235, 33

D'Isanto A., Polsterer K. L., 2018, Astronomy & Astrophysics, 609, A111

Edgeworth K. E., 1943, Journal of the British Astronomical Association, 53, 181

Einstein A., 1936, Science, 84, 506

Eisenstein D. J., et al., 2005, The Astrophysical Journal, 633, 560

Eisenstein D. J., et al., 2011, The Astronomical Journal, 142, 72

Euclid Collaboration et al., 2020, Astronomy & Astrophysics, 644, A31

Fawcett T., 2006, Pattern Recognition Letters, 27, 861

Feldmann R., et al., 2006, Monthly Notices of the Royal Astronomical Society, 372, 565

Fienga A., Laskar J., Manche H., Gastineau M., 2016, Astronomy & Astrophysics, 587, L8

Firth A. E., Lahav O., Somerville R. S., 2003, Monthly Notices of the Royal Astronomical Society, 339, 1195

Fixsen D. J., 2009, The Astrophysical Journal, 707, 916

Flaugher B., et al., 2015, The Astronomical Journal, 150, 150

Freund Y., Schapire R. E., 1997, Journal of Computer and System Sciences, 55, 119

Friedman J. H., 2002, Computational Statistics & Data Analysis, 38, 367

Galle J., 1846, Monthly Notices of the Royal Astronomical Society, 7, 153

Garg A., Tai K., 2013, International Journal of Modelling, Identification and Control, 18, 295

Gerdes D. W., Sypniewski A. J., McKay T. A., Hao J., Weis M. R., Wechsler R. H., Busha M. T., 2010, The Astrophysical Journal, 715, 823

Gerdes D. W., et al., 2016, The Astronomical Journal, 151, 39

Gerdes D., et al., 2017, The Astrophysical Journal Letters, 839, L15

Geurts P., Ernst D., Wehenkel L., 2006, Machine Learning, 63, 3

Goldstein R., 1993, Conditioning diagnostics: Collinearity and weak data in regression

Gomes R., Levison H. F., Tsiganis K., Morbidelli A., 2005, Nature, 435, 466

Gomes R. S., Fernández J. A., Gallardo T., Brunini A., 2008, The Solar System Beyond Neptune, pp 259–273

Gomes R., Deienno R., Morbidelli A., 2016, The Astronomical Journal, 153, 27

Graff P., Feroz F., Hobson M. P., Lasenby A., 2014, Monthly Notices of the Royal Astronomical Society, 441, 1741

Graham M. L., Connolly A. J., Ivezić Ž., Schmidt S. J., Jones R. L., Jurić M., Daniel S. F., Yoachim P., 2018, The Astronomical Journal, 155, 1

Gunn J. E., et al., 2006, The Astronomical Journal, 131, 2332

Hahn J. M., Malhotra R., 2005, The Astronomical Journal, 130, 2392

Hale T., et al., 2021, Nature Human Behaviour, 5, 529

Hamilton S., 2019, PhD thesis, University of Michigan

Hamilton S. J., et al., 2021, in prep

Hand D. J., Yu K., 2001, International Statistical Review, 69, 385

Harris C. R., et al., 2020, Nature, 585, 357

Harvey A., Peters S., 1990, Journal of Forecasting, 9, 89

Hastie T., Tibshirani R., Friedman J., 2009, in , The elements of statistical learning. Springer

Henghes B., et al., 2020, Publications of the Astronomical Society of the Pacific, 133, 014501

Henghes B., Pettitt C., Thiyagalingam J., Hey T., Lahav O., 2021, Monthly Notices of the Royal Astronomical Society, 505, 4847

Heymans C., et al., 2021, Astronomy & Astrophysics, 646, A140

Hildebrandt H., et al., 2010, Astronomy & Astrophysics, 523, A31

Hildebrandt H., et al., 2021, Astronomy & Astrophysics, 647, A124

Ho T. K., 1995, in Proceedings of 3rd International Conference on Document Analysis and Recognition. pp 278–282 vol.1

Holman M. J., et al., 2018, The Astrophysical Journal Letters, 855, L6

Holz D. E., Hughes S. A., 2005, The Astrophysical Journal, 629, 15

Horner J., Evans N. W., Bailey M. E., Asher D. J., 2003, Monthly Notices of the Royal Astronomical Society, 343, 1057

Hoyle B., 2016, Astronomy and Computing, 16, 34

Hubble E. P., 1929, The Astrophysical Journal, 69, 103

Hubel D. H., Wiesel T. N., 1968, The Journal of physiology, 195, 215

Huchra J., Davis M., Latham D., Tonry J., 1983, The Astrophysical Journal Supplement Series, 52, 89

Ida S., Bryden G., Lin D., Tanaka H., 2000, The Astrophysical Journal, 534, 428

Ilbert O., et al., 2006, Astronomy & Astrophysics, 457, 841

Ivezić Ž., et al., 2019, The Astrophysical Journal, 873

Jackson J. C., 1972, Monthly Notices of the Royal Astronomical Society, 156, 1P

Jain A., Duin R., Mao J., 2000, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22, 4

Johnson R., Zhang T., 2013, in Burges C. J. C., Bottou L., Welling M., Ghahramani Z., Weinberger K. Q., eds, , Advances in Neural Information Processing Systems 26. Curran Associates, Inc., pp 315–323

Kaiser N., 1987, Monthly Notices of the Royal Astronomical Society, 227, 1

Kaiser N., et al., 2002, in Survey and Other Telescope Technologies and Discoveries. pp 154–164

Kant I., 1755, Universal Natural History and Theory of Heaven

Kearns M., Valiant L., 1994, Journal of the ACM (JACM), 41, 67

Kessler R., et al., 2015, The Astronomical Journal, 150, 172

Khain T., et al., 2018, The Astronomical Journal, 156, 273

Khain T., et al., 2020, The Astronomical Journal, 159, 133

Kirk D., 2007, in Proceedings of the 6th International Symposium on Memory Management. ISMM '07. Association for Computing Machinery, p. 103–104

Kohavi R., 1995, in Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2. IJCAI'95. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 1137–1143

Kozai Y., 1962, The Astronomical Journal, 67, 591

Krasinsky G., Pitjeva E., Vasilyev M., Yagudina E., 2002, Icarus, 158, 98

Kuiper G. P., 1951, Proceedings of the National Academy of Sciences of the United States of America, 37, 1

Laplace P. S., 1799, Traité de mécanique céleste. de l'Imprimerie de Crapelet

Lauer T. R., et al., 2021, The Astrophysical Journal, 906, 77

Laureijs R., et al., 2011, arXiv preprint arXiv:1110.3193

Le Verrier U.-J.-J., 1839, CRAS, 9, 370

Leavitt H. S., 1908, Annals of Harvard College Observatory, 60, 87

Leistedt B., Hogg D. W., 2017, The Astronomical Journal, 838, 5

Lidov M., 1961, Iskusstviennye Sputniki Zemli, 8, 5

Lin H. W., et al., 2019, Icarus, 321, 426

Loh E., Spillar E., 1986, The Astrophysical Journal, 303, 154

Ma Z., Hu W., Huterer D., 2006, The Astrophysical Journal, 636, 21

Mandelbaum R., et al., 2008, Monthly Notices of the Royal Astronomical Society, 386, 781

Mandelbaum R., et al., 2018, arXiv preprint arXiv:1809.01669

Manning C. D., Raghavan P., Schutze H., 2008, Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA

Martini P., et al., 2018, in Ground-based and Airborne Instrumentation for Astronomy VII. SPIE, pp 410 – 420

McCulloch W. S., Pitts W., 1943, The bulletin of mathematical biophysics, 5, 115

Morbidelli A., Levison H. F., Tsiganis K., Gomes R., 2005, Nature, 435, 462

Myles J., et al., 2021, Monthly Notices of the Royal Astronomical Society, 505, 4249

Nair V., Hinton G. E., 2010, in Proceedings of the 27th International Conference on International Conference on Machine Learning. ICML'10. p. 807–814

Neilsen E., Bernstein G., Gruendl R., Kent S., 2016, Technical report, Limiting magnitude, τ, teff, and image quality in DES Year 1. Fermi National Accelerator Lab.(FNAL), Batavia, IL, USA

Nesvornỳ D., 2015, The Astronomical Journal, 150, 73

Nesvornỳ D., Morbidelli A., 2012, The Astronomical Journal, 144, 117

Nesvornỳ D., Vokrouhlickỳ D., 2016, The Astrophysical Journal, 825, 94

Newton I., 1672, Philosophical Transactions of the Royal Society of London, 6, 3075

Parker R. J., Lichtenberg T., Quanz S. P., 2017, Monthly Notices of the Royal Astronomical Society: Letters, 472, L75

Pasquet J., Bertin E., Treyer M., Arnouts S., Fouchez D., 2019, Astronomy & Astrophysics, 621, A26

Pearson K., 1901, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2, 559

Pedregosa F., et al., 2011, The Journal of Machine Learning Research, 12, 2825

Penzias A. A., Wilson R. W., 1965, The Astrophysical Journal, 142, 419

Perlmutter S., et al., 1999, The Astrophysical Journal, 517, 565

Petrosian V., 1976, The Astrophysical Journal, 209, L1

Pitjeva E., Pitjev N., 2018, Celestial Mechanics and Dynamical Astronomy, 130, 1

Planck Collaboration 2020, Astronomy & Astrophysics, 641

Puschell J., Owen F., Laing R., 1982, The Astrophysical Journal, 257, L57

Quinlan J. R., 1986, Machine learning, 1, 81

Quinlan J. R., 1993, C 4.5: Programs for machine learning

Resnick P., Varian H. R., 1997, Communications of the ACM, 40, 56

Riess A. G., et al., 1998, The Astrophysical Journal, 116, 1009

Rumelhart D. E., Hinton G. E., Williams R. J., 1986, nature, 323, 533

Sadeh I., Abdalla F. B., Lahav O., 2016, Publications of the Astronomical Society of the Pacific, 128, 104502

Samuel A. L., 1959, IBM Journal of Research and Development, 3, 210

Sánchez C., et al., 2014, Monthly Notices of the Royal Astronomical Society, 445, 1482

Schmidt S. J., et al., 2020, Monthly Notices of the Royal Astronomical Society, 499, 1587

Scholtz J., Unwin J., 2020, Phys. Rev. Lett., 125, 051103

Schuldt S., Suyu S., Cañameras R., Taubenberger S., Meinhard T., Leal-Taixé L., Hsieh B., 2020, arXiv preprint arXiv:2011.12312

Schutz B. F., 1986, Nature, 323, 310

Sejnowski T. J., 2018, The deep learning revolution. Mit Press

Shankman C., et al., 2017, The Astronomical Journal, 154, 50

Sheppard S., Trujillo C., Tholen D., Kaib N., 2019, The Astronomical Journal, 157

Snoek J., Larochelle H., Adams R. P., 2012, in , Advances in Neural Information Processing Systems 25. Curran Associates, Inc., pp 2951–2959

Soares-Santos M., et al., 2019, The Astrophysical Journal Letters, 876, L7

Soo J. Y. H., et al., 2017, Monthly Notices of the Royal Astronomical Society, 475, 3613

Spergel D., et al., 2015, arXiv preprint arXiv:1503.03757

Springel V., 2005, Monthly notices of the royal astronomical society, 364, 1105

Storrie-Lombardi M., Lahav O., Sodre Jr L., Storrie-Lombardi L., 1992, Monthly Notices of the Royal Astronomical Society, 259, 8P

Szegedy C., et al., 2015, in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp 1–9

Szegedy C., Vanhoucke V., Ioffe S., Shlens J., Wojna Z., 2016, in Proceedings of the IEEE conference on computer vision and pattern recognition. pp 2818–2826

Taylor S. J., Letham B., 2018, The American Statistician, 72, 37

Tombaugh C., 1946, Astronomical Society of the Pacific Leaflets, 5, 73

Trujillo C. A., Sheppard S. S., 2014, Nature, 507

Tsiganis K., Gomes R., Morbidelli A., Levison H. F., 2005, Nature, 435, 459

Turner H. H., 1912, The Great Star Map: Being a General Account of the International Project Known as the Astrographic Chart

Tyson J., Wittman D., Hennawi J., Spergelb D., 2003, Nuclear Physics B-Proceedings Supplements, 124, 21

Urban S. E., Corbin T. E., 1998, Sky and Telescope, 95, 40

Van Geffen J., Van Weele M., Allaart M., 2017, TEMIS UV index and UV dose operational data products, version 2, Dataset

Xu R., Rahmandad H., Gupta M., DiGennaro C., Ghaffarzadegan N., Amini H., Jalali M. S., 2020, Social Science Research Network 3593879

Yao Y., Pan J., Liu Z., Meng X., Wang W., Kan H., Wang W., 2020, European Respiratory Journal, 55

York D. G., et al., 2000, The Astronomical Journal, 120, 1579

Yudistira N., Sumitro S. B., Nahas A., Riama N. F., 2020, medRxiv preprint medRxiv:2020.04.30.20086983

de Jong J. T., Kleijn G. A. V., Kuijken K. H., Valentijn E. A., 2013, Experimental Astronomy, 35, 25 de la Fuente Marcos C., de la Fuente Marcos R., 2014, Monthly Notices of the Royal Astronomical Society, 443, L59