Machine Learning in Astronomy

by

Lise du Buisson

Minor Dissertation presented in partial fulfilment of the requirements for the degree of Master of Science

in the

Department of Mathematics and Applied Mathematics

University of Cape Town

May 2015

Supervisor: Professor Bruce A. Bassett

Declaration of Authorship

I, Lise du Buisson, know the meaning of plagiarism and declare that all of the work in the dissertation titled "Machine Learning in Astronomy", save for that which is properly acknowledged, is my own.

It should be noted that a paper entitled "Machine Learning Classification of SDSS Transient Survey Images" by du Buisson et al. [1], based on the work in this thesis, has been submitted to the *Monthly Notices of the Royal Astronomical Society (MNRAS)* journal. The current version of the paper can be found at `http://arxiv.org/abs/1407.4118`. Some sections from the paper have been modified for the purposes of using it in this thesis, and Figures 2.5, 3.6, 4.3, 5.1, 5.2, 5.3 and 5.4 as well as Tables 3.2, 5.1, 5.7, 5.9, 5.10, 5.11, 5.12 and C.1 have been produced both for use in the paper and this thesis, and so appear in both.

Signed by candidate

Signed: Signature removed

Date: 21 May 2015

"There are worlds out there where the sky is burning, and the sea's asleep, and the rivers dream; people made of smoke and cities made of song. Somewhere there's danger, somewhere there's injustice, and somewhere else the tea's getting cold. Come on, Ace. We've got work to do."

The Seventh Doctor, *Doctor Who*

UNIVERSITY OF CAPE TOWN

Abstract

Faculty of Science
Department of Mathematics and Applied Mathematics

Master of Science

by Lise du Buisson

The search to find answers to the deepest questions we have about the Universe has fueled the collection of data for ever larger volumes of our cosmos. The field of supernova cosmology, for example, is seeing continuous development with upcoming surveys set to produce a vast amount of data that will require new statistical inference and machine learning techniques for processing and analysis. Distinguishing between real objects and artefacts is one of the first steps in any transient science pipeline and, currently, is still carried out by humans - often leading to hand scanners having to sort hundreds or thousands of images per night. This is a time-consuming activity introducing human biases that are extremely hard to characterise. To succeed in the objectives of future transient surveys, the successful substitution of human hand scanners with machine learning techniques for the purpose of this artefact-transient classification therefore represents a vital frontier. In this thesis we test various machine learning algorithms and show that many of them can match the human hand scanner performance in classifying transient difference g, r and i-band imaging data from the SDSS-II SN Survey into real objects and artefacts. Using principal component analysis and linear discriminant analysis, we construct a grand total of 56 feature sets with which to train, optimise and test a Minimum Error Classifier (MEC), a naïve Bayes classifier, a k-Nearest Neighbours (kNN) algorithm, a Support Vector Machine (SVM) and the SkyNet artificial neural network. We find that kNN is the best-performing classifier with an accuracy of 89% and a recall of 90%. Three of our classifiers (kNN, SkyNet and MEC) match the human recall performance of $\sim 96\%$ on the fake SNe injected into the SDSS pipeline, accomplishing our goal. We also show that none of our best-performing classifiers are highly correlated in their classifications, an indication that an ensemble classifier might yield improved results. Extending this research by further making use of multiple epoch data, object histories and host galaxy information can lead to further improvements.

Acknowledgements

First and foremost, I thank my supervisor, Bruce Bassett - for readily providing me with advice and support whenever I was in need of it, and for setting a great example in both a personal and academic capacity. For making me look up and remember the bigger picture every once in a while - it sometimes does wonders to remember that soon we'll all be dead and we should live our lives doing what we want and should to be happy.

I thoroughly enjoyed collaborating with Navin Sivanandam, "my" postdoc, who were always willing to lend a friendly hand when I needed explanations and who made the environment fun with his interesting and unique take on life. Thanks also to Mathew Smith, another collaborator, for valuable advice and for supplying the data for my project.

This year would have been a lot less fun without Michelle, with whom I successfully created one of the world's most counter-productive office environments, and to whom I am extremely grateful for lending a kind ear in difficult times. My other office-mates, friends and colleagues gave AIMS the welcoming and warm character I will remember it by. They are Eli and Ermias (who had to put up with me and Michelle), Gilad, Kai, Rene, Mark and Iggy. Mika, Amidou and Ayman made long nights in the computer laboratory anything but tedious, and didn't hesitate to make me feel at home during those first days of NASSP when I was a bit dazed and confused.

I would not have been able to complete my masters were it not for the love and support from a few other very important people. My mother and father, for their love and support for anything and everything I do; my brother, Johan, for his friendship and our honest discussions that I value so much, and for all the tea and coffee; Nathan, for being a true friend - whose discussions on life, liberty and the pursuit of happiness always left me feeling free of all chains, and who braved the early morning traffic back to Stellenbosch many times for me during those first months; Nadeem, who I journeyed with for a very special few years that I will always cherish. And then, Filipe, who made this last year truly remarkable. For standing by my side through the good times and the bad, for arranging surprise visits from the UK, for proof-reading my thesis without complaining, for introducing me to Doctor Who and for being the most perfect adventure companion that I could've asked for.

Formal Acknowledgements

I acknowledge funding from the Square Kilometre Array (SKA) Project and the National Research Foundation (NRF) for the duration of my postgraduate studies, and support and resources from the African Institute for Mathematical Sciences (AIMS). I further thank Johan A. du Preez and Ben Herbst for their insights during the initial phases of the project, and the University College London (UCL) Physics and Astronomy Department for the use of their computer cluster, Splinter.

Funding for the SDSS and SDSS-II has been provided by the Alfred P. Sloan Foundation, the Participating Institutions, the National Science Foundation, the U.S. Department of Energy, the National Aeronautics and Space Administration, the Japanese Monbukagakusho, the Max Planck Society, and the Higher Education Funding Council for England. The SDSS Web Site is http://www.sdss.org.

The SDSS is managed by the Astrophysical Research Consortium for the Participating Institutions. The Participating Institutions are the American Museum of Natural History, Astrophysical Institute Potsdam, University of Basel, Cambridge University, Case Western Reserve University, University of Chicago, Drexel University, Fermilab, the Institute for Advanced Study, the Japan Participation Group, Johns Hopkins University, the Joint Institute for Nuclear Astrophysics, the Kavli Institute for Particle Astrophysics and Cosmology, the Korean Scientist Group, the Chinese Academy of Sciences (LAMOST), Los Alamos National Laboratory, the Max-Planck-Institute for Astronomy (MPIA), the Max-Planck-Institute for Astrophysics (MPA), New Mexico State University, Ohio State University, University of Pittsburgh, University of Portsmouth, Princeton University, the United States Naval Observatory, and the University of Washington.

Contents

List of Figures

List of Tables

Abbreviations

AGN	**A**ctive **G**alactic **N**uclei
AI	**A**rtificial **I**ntelligence
ANN	**A**rtificial **N**eural **N**etwork
AUC	**A**rea **U**nder **C**urve
DES	**D**ark **E**nergy **S**urvey
FLD	**F**isher's **L**inear **D**iscriminant
FPR	**F**alse **P**ositive **R**ate
KKT	**K**arush-**K**uhn-**T**ucker
kNN	k-**N**earest **N**eighbours
LDA	**L**inear **D**iscriminant **A**nalysis
LSST	**L**arge **S**ynoptic **S**urvey **T**elescope
MEC	**M**inimum **E**rror **C**lassification
NB	**N**aive **B**ayes Classifier
PCA	**P**rincipal **C**omponent **A**nalysis
Pan-STARRS	**P**anoramic **S**urvey **T**elescope & **R**apid **R**esponse **S**ystem
PTF	**P**alomar **T**ransient **F**actory
ROC	**R**eceiver **O**perator **C**haracteristic
RNN	**R**ecurrent **N**eural **N**etwork
SDSS	**S**loan **D**igital **S**ky **S**urvey
SLSN	**S**uper**l**uminous **S**uper**n**ova
SN	**S**uper**n**ova
SNR	**S**ignal to **N**oise **R**atio
SVM	**S**upport **V**ector **M**achine
TPR	**T**rue **P**ositive **R**ate

Physical Constants

Solar Mass	$\mathrm{M}_\odot$	$=$	$(1.98855 \pm 0.00025) \times 10^{30}$ kg
Hubble Constant	H_0	$=$	67.80 ± 0.77 km s^{-1} Mpc^{-1}
Parsec	pc	$\approx$	3.1×10^{16} m
Speed of Light	c	$=$	2.99792458×10^{8} m s^{-1}

To my parents

Chapter 1

The History of Machine Learning

> *For seeing life is but a motion of limbs, the beginning whereof is in some principle part within, why may we not say that all automata (engines that move themselves by springs and wheels as doth a watch) have an artifical life? For what is the heart, but a spring; and the nerves, but so many strings; and the joints, but so many wheels, giving motion to the whole body...*
>
> – Thomas Hobbes, *The Leviathan*

1.1 Introduction

The field of machine learning concerns itself with the construction of computer programs that automatically improve their performance with experience. It draws on many ideas and concepts from many fields, including philosophy, biology, mathematics, statistics, control theory, computational complexity and logic, among others. It plays an increasingly important role in our modern society and finds applications in diverse areas such as computer vision, natural language processing, search engines, medical diagnosis, stock market analysis, pattern and speech recognition, robot locomotion and information retrieval.

It is impossible to have a clear idea of what machine learning is and where it comes from without looking at its slightly older and wiser uncle, Artificial Intelligence (AI), from which it grew. Although AI lacks a clear-cut and agreed-upon definition, it can broadly be seen as the field devoted to the study and design of intelligent agents, where intelligence can be taken as that attribute an entity requires in order for it to function properly and with foresight in its environment. According to this definition, intelligence lends itself to many things: human beings, animals, and certain machines. They all have varying degrees of intelligence, and cover different ranges of whom many are overlapping and have unclear boundaries. Because of a lack of sharp discontinuities between

these ranges, it is better to have a broad view of what intelligence is and what AI, therefore, constitutes. [2]

This chapter will present a short history of AI and machine learning in order to orientate the reader, and to give some insight as to the origins and importance of these two intertwined, and often indistinguishable, fields. It will outline humanity's AI dreams and achievements, look at the ideas and factors that influenced the field's evolution over the years, and discuss machine learning and its place in the broader AI spectrum from which it branches. It should be noted that the primary source of information for this section was the book by Russell and Norvig [3], and that where not indicated otherwise, it should be assumed that information given was obtained from this source.

1.2 AI and Machine Learning: A Modern History

As mentioned above, AI and machine learning can in many cases be seen as two quite indistinguishable fields, sharing many methods and ideas - especially during their early years. We therefore detail the development of both these fields, as it is impossible to cover one field without involving the other. We will often use the terms "AI" and "machine learning" interchangeably, and will cover the period of development from 1943 up to the present. It should be noted, of course, that this history is by no means exhaustive, and that some facts, ideas and people of importance had to be omitted for the sake of brevity.

1.2.1 The Gestation Period (1943-1955)

The first work now generally recognised as modern AI and machine learning was done by Warren McCulloch and Walter Pitts in 1943 [4]. They put forth a network of artificial neurons, where each neuron is either "on" or "off", a switch to "on" taking place when a neuron is stimulated by sufficiently many neighbouring neurons. They showed that any given function could be calculated by some neuron network and that all logical operators could be expressed with simple networks. More than that, they suggested that networks can be made to learn - and indeed, in 1949 Donald Hebb developed *Hebbian Learning* [5], showing that by using an updating rule to modify the strengths of the connections between neurons, learning was indeed possible. These discoveries put a start to the field of Artificial Neural Networks (ANNs), a family of statistical machine learning algorithms widely used today (ANNs are discussed in section 2.2.4). Soon after, in 1951, Marvin Minsky and Dean Edmonds developed the first ever neural network computer, called the SNARC (Stochastic Neural Analogue Reinforcement Computer). It consisted of 40 neurons, employed Hebbian Learning, and was able to learn some simple concepts [6, 7].

Amongst other early work considered to be AI, Alan Turing's vision was of particular importance. In a paper he wrote in 1950 [8], he introduced the famous Turing Test[1] and the concept of machine learning.

1.2.2 The Dawn of Artificial Intelligence (1956)

In 1956, John McCarthy, Marvin Minsky, Nathaniel Rochester and Claude Shannon organised a workshop at Dartmouth College for researchers interested in neural nets, automata theory[2] and the study of intelligence - the workshop's proposal [9] includes the first official use of the term *artificial intelligence*. The 10 attendees included Arthur Samuel (IBM[3]), Trenchard More (Princeton) and Oliver Selfridge and Ray Solomonoff (MIT[4]). Herbert Simon and Allen Newell, two attendees from Carnegie Tech[5], already had a reasoning program named the Logic Theorist (LT) [10] which, soon after the workshop, was able to prove most theorems in Chapter 2 of *Principia Mathematica* [11] - LT even found a shorter proof for one of the theorems.

The Dartmouth workshop itself did not lead to any breakthroughs, but it introduced all the major figures in the field to one another. The attendees, their colleagues and their students at MIT, IBM, CMU and Stanford University would dominate the field for the coming 20 years. The workshop also established AI as a separate field, with goals distinct from fields like control theory, operations research, decision theory and mathematics in that it strives to address issues like the duplication of human faculties (e.g. self-improvement, creativity, and language use) and in the process attempts to create machines that function autonomously.

1.2.3 Early Days and Expectations (1952-1969)

Simon and Newell followed up their earlier successes by developing the General Problem Solver (GPS) [12] which, unlike LT, was programmed from the start to emulate protocols of human problem-solving. GPS and the cognition model programs that followed led to their formulation of the *physical symbol system* hypothesis [13, 14], stating that a system, whether it be human or machine, displaying intelligence must function through the manipulation of data structures consisting of symbols.

[1] Highly influential and widely criticised, the Turing test examines a machine's ability to display intelligent behaviour indistinguishable from a human's. In the original setting, a human judge has conversations with a machine and a human, with all participants separated from each other. If the judge can't distinguish between the human and the machine, the machine has passed the Turing test. The Russian chatter bot Eugene Goostman, developed by Vladimir Veselov et al., came closest yet to passing the Turing test in 2014 by fooling 33 % of the judges at the "Turing2014: Can Machines Think?" event (visit `http://www.robolaw.eu/news.htm#turing2014`).

[2] The study of self-operating machines and the computational problems they can solve.

[3] International Business Machines Corporation.

[4] Massachusetts Institute of Technology.

[5] Now Carnegie Mellon University (CMU).

At IBM development also went forwards, with some of the earliest AI programs having been created by Nathaniel Rochester and his colleagues. Herbert Gelernter developed the Geometry Theorem Solver in 1959 [15], a program that was capable of proving theorems that many students of mathematics would find challenging, and in 1952 Arthur Samuel (using decision-theoretic techniques) wrote checkers programs [16, 17] that quickly learned how to play a better game than he could himself, disproving the idea that computers could only do what they were told to. Samuel's efforts were among the first in what would later become the field of machine learning. In 1958 John McCarthy, now at MIT, developed the Lisp programming language and invented, together with others at MIT, time sharing[6]. In the same year he also described a hypothetical program called the Advice Taker [18], seen by some as the first complete AI system. It used knowledge to find solutions to problems, but unlike LT and the Geometry Theorem Solver it made use of general world knowledge - it was designed to accept new axioms during the course of its operations, allowing it to become competent in new areas. The program therefore made use of the key principles of reasoning and knowledge representation: a formal representation of the world together with the ability to manipulate it.

In the 1960's Minsky, now also at MIT, supervised students who studied problems that seemed to require intelligence in order to be solved. These problems were called *microworlds*, the most famous one having been the "blocks world", comprising of a set of blocks on a table-top operated by a robot hand. It was home to David Huffman's vision project [19] and Terry Winograd's natural-language-understanding program, SHRDLU [20], among others.

Meanwhile, work that built on McCulloch and Pitts's neural networks also flourished - Winograd and Cowan [21] showed that an individual concept could be represented by a large number of collective elements and Hebb's learning methods were improved by Bernie Widrow [22, 23] and by Frank Rosenblatt [24], who introduced perceptrons[7]. The perceptron convergence theorem [25], published by Block et al. in 1962, states that a learning method can modify a perceptron's connection strengths to match any input data, if such a match exists.

1.2.4 Reality Strikes (1966-1973)

The promising performance of early AI solutions to elementary problems led to a general overconfidence in the community of what was achievable at the time. In most cases these early AI systems failed when applied to more diverse or more complex problems. This was due to a few difficulties.

[6] Time sharing is the process by which multiple users having different programs can interact with the central processing unit (CPU) of a computer nearly simultaneously. It represents a very important technological shift in computing history.

[7] An algorithm for transforming an input into one of several possible non-binary outputs. Perceptrons are discussed in section 2.2.4.1.

The first was that, because most early language-processing AI programs didn't know anything about their subject matter, they worked by using simple syntactic manipulations. An example is early machine translation efforts, where it was initially thought that word replacements and basic syntactical transformations based on the different languages' grammar would be sufficient to preserve exact sentence meanings. The truth is that background knowledge is required for accurate translation, so that language phenomena like ambiguity doesn't stand in the way of determining a sentence's content. Although it has found widespread use, machine translation remains imperfect until this day.

The second difficulty arose from the intractable nature of most problems AI was trying to solve. Many early AI algorithms solved problems by attempting steps in different combinations until they arrived at the solution - this worked due to the fact that many problems was of a simple nature requiring short solution sequences. Before the advent of the field of computational complexity, it was assumed that larger problems would simply require larger memories and faster hardware. It was soon realised to be a misconception, an example being the dampened optimism accompanying the development of resolution theorem proving when programs couldn't prove theorems containing more than a dozen or so facts. This assumption was not limited to problem-solving programs - early development in genetic algorithms [26, 27] were grounded on the correct idea that by making a series of mutations to a program, one could evolve it to have a good performance on any specific task. This was carried out by trying random mutations and keeping the useful ones. During early development almost no progress was made, despite thousands of CPU hours - modern genetic algorithms, however, have shown more success by using better representations and plays an important role in modern approaches to machine learning.

Lastly, problems arose because of the fact that basic AI structures that was being used for the generation of intelligence had some fundamental limitations. An example of this is the book "Perceptrons" by Minsky and Papert in 1969 [28], in which they proved that although perceptrons have the ability to learn anything they could represent, they could represent very little, causing research funding for neural networks to become almost non-existent. The backpropagation algorithms[8] for multilayer networks that resulted in a resurgence of neural network research in the 1980s was, ironically, discovered in 1969 by Bryson and Ho [29].

1.2.5 Knowledge-Based Systems (1969-1979) and the AI Industry (1980-present)

The first decade of AI research was dominated by approaches using general-purpose search methods stringing together simple reasoning steps in order to arrive at complete solutions. These algorithms have been called *weak methods* due to their inability to scale up to larger or more complicated problems. Alternatively, methods using powerful, domain-specific knowledge can be used - this

[8] An abbreviation of "backward propagation of errors", this is a common method used for the training of ANNs. See section B.2 for more details.

allows larger reasoning steps to be taken and results in better performance on typically occurring problems in narrower areas of expertise.

The DENDRAL program, developed by Buchanan et al. in 1969 [30], is an example of such an *expert system*. DENDRAL was used to help tackle the problem of inferring molecular structure using spectrometer information. Its significance lies in the fact that it was the first successful knowledge-intensive program - it made use of many special-purpose rules to perform well on the problem.

Shortly after, Feigenbaum and his colleagues at Stanford started the Heuristic Programming Project (HPP) in order to determine to what extent expert systems could be used for other domains of human expertise. In the area of medical diagnosis, Feigenbaum, Shortliffe and Buchanan created the MYCIN [31, 32], a program running on 450 rules that could diagnose blood infections with a performance equivalent to that of some human experts in the field. Expert systems also made an appearance in the field of natural language processing. Terry Winograd's program SHRDLU (mentioned earlier as part of the blocks microworld) overcame ambiguity and understood pronoun preferences, although this was mainly the result of it being developed for one very specific area - the blocks world. Its dependence on syntactical analysis caused some of the same problems as mentioned before. Others suggested that a robust understanding of language would require general world knowledge, and corresponding methods for the usage of such knowledge. One of these researchers was Robert Schank who, with his students, built programs [33–36] all with the task of understanding natural language - with the emphasis more on representing and reasoning with the knowledge necessary for comprehending natural language than on language itself.

Due to the commercialisation of expert systems, the AI industry grew from a worth of millions of dollars in 1980 to a worth of billions in 1988. Soon after, though, companies became unprofitable as they couldn't deliver on their extravagant promises. People lost faith in AI and the following years saw a corresponding drop in funding for AI research and projects - a time now referred to as the "AI winter". This draught was, in part, also due to the nature of AI - being founded, in part, as a field in rebellion against the limitations of existing fields like statistics and control theory, it has not yet reached a point of integration with these helpful approaches.

1.2.6 The Rise of Machine Learning (1980-present)

Up to this point I have (mostly) used the term "AI" as an overarching name for AI and all of its constituents - seeing as AI was still a forming field and boundaries between it and other fields not clear at all times, this was acceptable. We have now, however, reached a point in history where the distinguishing line between machine learning and its parent field AI becomes more apparent.

In 1980, the first machine learning workshop held at CMU identified a group of researchers with a shared interest in computational approaches to learning - at the time, the parent field of AI was showing little interest in learning-related issues, choosing instead to concentrate on logical, knowledge-based approaches such as those required for the design of expert systems. The use of statistics in AI fell out of favour, and though work on the symbolic/knowledge-based approach to learning did continue, its statistical line of research was now outside the domain of AI, in pattern recognition and information retrieval. Machine learning changed its former goal of achieving artificial intelligence to that of solving problems of a more practical nature. It shifted its focus away from the symbolic approaches it inherited from AI, and drifted towards methods borrowed from statistics, pattern recognition and probability theory. [37] It should be noted that recent advances in AI, however, have seen it re-establishing common ground with other disciplines such that the field is now commonly referred to as "Modern AI" - machine learning and AI are once again very integrated.

During the mid-1980s, the backpropagation algorithm originally discovered by Bryson and Ho in 1969 [29] was reinvented by four different research groups. It was used for many problems in computer science, and the collection of the results, entitled "Parallel Distributed Processing", by Rumelhart and McClelland in 1986 [38] caused great enthusiasm and set the stage for neural network research to flourish once more - today ANNs are valuable machine learning techniques.

Machine learning, by now a well-defined research field, began to flourish in the 1990s. Whereas before 1980 it was mainly represented by neural network methods, it quickly broadened its arsenal to include (to name but a few) memory-based learning (e.g. the k-nearest neighbours method - section 2.2.2), case-based reasoning[9], decision tree learning[10], support vector machines (section 2.2.3), unsupervised and reinforcement learning methods (section 2.1.2) and deep learning techniques[11] [2]. The success of machine learning has also been helped along by the advent of large data sets in the mid 1990's to early 2000s. The more data available for training, the more likely it becomes that learning algorithms will be able to learn successfully from it. This also led to the relaxed optimisation of the learning rate of algorithms - where before algorithms had to use small datasets and therefore had to have high learning rates to be successful, they now have the luxury of vast datasets that relaxes the emphasis on efficient learning. The more data, the better - an aspect that made machine learning techniques ideal for data mining[12] applications [37].

[9] Broadly speaking, the process by which the solutions to past problems are used to find solutions to similar new ones.

[10] Learning that uses a decision tree as a predictive model to map observations representing a data instance to conclusions about the item's nature.

[11] Algorithms modelling high-level data abstractions using models consisting of numerous nonlinear functions.

[12] The computational process by which patterns in large data sets are discovered, usually using methods from AI, statistics and machine learning.

Today, machine learning applications are widely used all around us. Learning techniques classify over a billion messages as junk e-mail every day [39, 40] and self-driving vehicles are making headlines: a robotic vehicle named Stanley (developed by Stanford researchers) succeeded in winning the 2005 DARPA Grand Challenge[13] [41], the autonomous vehicle called Boss (developed by CMU) won the DARPA Urban Challenge[14] the next year [42] and the "Google Self-Driving Car" project continues to improve on the design of cars that can function autonomously in city environments[15]. In game-playing, IBM's chess program named "Deep Blue" became the first computer program to beat a world champion (Garry Kasparov) in 1997 [43], and applications of machine learning algorithms in the areas of language translation, speech recognition, medical diagnosis, robot locomotion, data analysis, stock market analysis and software engineering abounds. This is of course not an exhaustive list - machine learning is practically everywhere around us, and the coming years will see the continued growth of the field.

1.3 Machine Learning Applications in Astronomy

Applications of machine learning in astronomy is of particular interest to this thesis, and so this section discusses the usage of these algorithms in different areas and for different problems in the field as a whole. This is by no means a complete overview of such work, and serves merely to give the reader an idea of the applicability of machine learning algorithms in the field. This section mentions various machine learning algorithms - the reader should note that neural networks (section 2.2.4), principal component analysis (section 2.3.1), k-nearest neighbours (section 2.2.2) and support vector machines (section 2.2.3) will be covered later in the thesis, as indicated. The work of Ball & Brunner [44] was used as a source of information for this section.

1.3.1 Classification of Objects

A well-studied application of machine learning in astronomy is that of *star-galaxy separation* - the large number of stars (appearing as point sources) and galaxies (appearing as extended sources) in photometric catalogues make this a problem requiring automation. Several algorithms, including neural networks [45–51] and decision trees [52, 53], have been used to tackle this problem, with most of these algorithms achieving a precision score of more than 95%[16], typically done using algorithm inputs of morphological parameters derived from the survey photometry.

A second problem that machine learning is often applied to is that of distinguishing between different *galaxy morphologies*. Galaxies come in many shapes and sizes, and knowledge of a particular

[13] http://www.darpa.mil/About/History/Archives.aspx
[14] http://archive.darpa.mil/grandchallenge/
[15] Google: http://www.google.com/about/company/ Google research: http://research.google.com/
[16] The fraction of relevant retrieved instances. See section 5.2 for a full description of the precision performance metric.

galaxy's morphology can give information as to its formation and evolution. Using machine learning algorithms, one can assign galaxy morphologies to images when measured parameters (such as morphological parameters or colour information) are available - using neural networks on low redshift[17] data, Lahav and his collaborators wrote a series of papers [54–59] carrying out the above, and found that it is possible to obtain a classification accuracy similar to that of human experts. Other studies have applied neural networks to higher redshift data [60], and morphological types have also been assigned to galaxies by neural networks using galaxy spectra as input [61]. Galaxy morphology at higher redshift is also studied by using data from the Hubble Deep Field[18] where galaxies are fainter, more distant, morphologically more peculiar and less evolved - three papers [62–64] use surface brightness and light profiles of galaxies as inputs to neural networks in order to classify them. Other classifications of galaxies have also been tackled with machine learning - numerous papers [65–68] applied principal component analysis directly in order to obtain the spectral classifications of galaxies.

Most of the electromagnetic radiation in the Universe is emitted by either stars or by the accretion disks around supermassive black holes in Active Galactic Nuclei[19] (AGN). In the case of quasars[20] for the latter, the central region of the galaxy can outshine the entire galaxy it resides in. Due to the fact that supermassive black holes are thought to be pervasive in larger galaxies, and because their intrinsic brightness can be influenced by the host galaxy's environment, AGN and quasars are important tools for understanding the evolution and formation of structure in the Universe. The identification of AGN and quasars in astronomical surveys is therefore an important and well-studied problem. Several people have used neural networks [69–71] and decision trees [72–74] for the selection of quasars from surveys, where many of the above studies combine multi-wavelength data. In a similar fashion, the selection and classification of AGN can be done - a paper by Zhao et al. [75] details the spectral classification of AGN using a k-nearest neighbours method.

1.3.2 Photometric Redshifts

Redshift estimation using photometric data is an area of study that has become more and more popular during the last few years. This reason for this is that, even though photometric redshifts are less accurate than the ones obtained with the use of spectra, the vast number of objects for which photometric measurements are available make up for the loss in individual accuracy through statistical noise suppression in ensemble calculations.

[17] See Eq. 3.10 for the definition of redshift.

[18] An image of a small region in the Ursa Major constellation, constructed from 342 exposures taken by the Hubble Space Telescope.

[19] These are galaxies with very bright and energetic central regions, caused by either the presence of a black hole or by star formation at their cores.

[20] Quasars (quasi-stellar radio sources) are the most distant and energetic AGNs found. The are extremely luminous, and emit electromagnetic energy giving them an appearance more star-like than galaxy-like - from there the term "quasi-stellar".

For normal galaxies at low redshift, the calculation of photometric redshifts is simple due to the typical galaxy spectrum break at 4000 Å- resulting in a relatively smooth colour change as a galaxy is redshifted with increasing distance. Studies using neural networks [76–80], support vector machines [81, 82], decision trees [83] and k-nearest neighbours [84] have been used for the determination of photometric redshifts for such galaxies.

The determination of photometric redshifts for AGN or quasars is more complicated, seeing as their spectra are dominated by bright and narrow emission lines which dominate the colour in broad photometric passbands. Machine learning applications to tackle the problem of photometric redshift determination for these objects include the use of k-nearest neighbours, neural networks and decision trees [85].

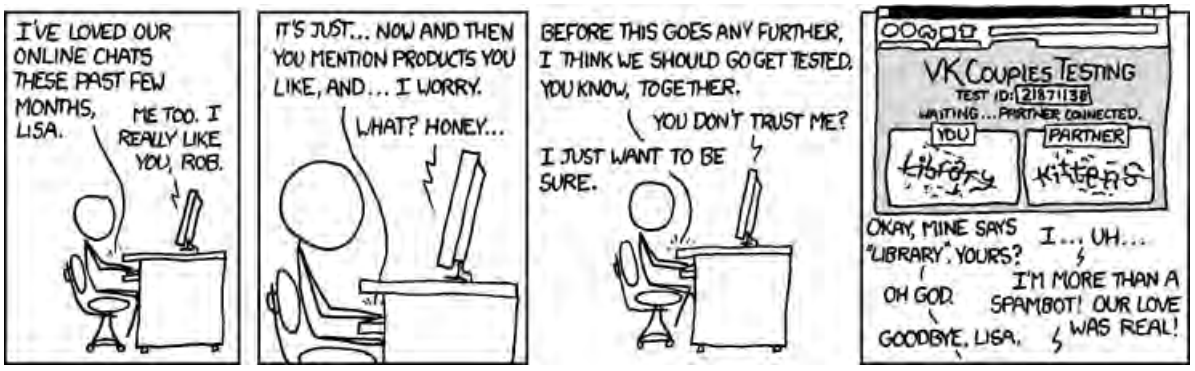
1.3.3 Other Applications

There are many other applications of machine learning in astronomy. Examples include that of Ramirez, Fuentes & Gulati [86], where physical parameters of stellar atmospheres are predicted using spectral indices. Here, a genetic algorithm is used for the selection of appropriate input features, after which the parameters are predicted using a k-nearest neighbours algorithm. More examples are the work of Mokiem et al. [87], where a fitting method based on a genetic algorithm is used to spectrally analyse early-type stars, and the work of Giridhar et al. [90], where metal-poor stars are identified with the help of a neural network. Other interesting examples include SkyNet, a generic neural network training algorithm developed by Graff et al. [88] and Source Extractor (SExtractor) [89], used for object detection.

Machine learning applied to time domain data is also an area in which a lot of work has been done. In this area, machine learning has been applied to quasar variability, gamma ray bursts, supernovae, novae, extrasolar planets, stellar proper motions, solar system planetary atmospheres, detection and classification of asteroids, transient lunar phenomena, and the interaction of the solar wind and the Earth's atmosphere. An overview of this can be found here [91].

Chapter 2

An Introduction to Machine Learning



– http://xkcd.com/632/

2.1 Introduction

Machine learning is concerned with creating computer algorithms that can automatically discover regularities in data and, using these regularities, improve their performance on a given task by learning through experience. This section will introduce some common terms and concepts that will be encountered throughout this thesis, after which it will proceed to discuss the three broad categories of machine learning applications. The two following sections in this chapter will discuss the various machine learning algorithms used later in the thesis (section 2.2) and the feature extraction techniques employed (section 2.3). It should be noted that, where not mentioned otherwise, the main sources of information for this chapter were the books by Bishop [92] and Mitchell [93].

2.1.1 Important Concepts

As an example of a typical machine learning problem, consider the handwritten digits shown in Fig. 2.1. Each digit (referred to more generally as a data *instance*) comprises of a 28×28 pixel image,

and can therefore be represented with a 784-element vector $\mathbf{x}$. These elements, characterising the data instance, are normally referred to as its *features*. The objective here is to design an algorithm that takes such a feature vector $\mathbf{x}$ as an input and produces as the output a classification as one of ten possible classes, with each class corresponding to one of the digits $0, \ldots, 9$. This problem is difficult because of the large variability found in people's handwriting. An attempt at a solution could be made by designing sets of rules or using heuristics in order to distinguish digits from one another - however, such methods often lead to an infeasible amount of rules and exceptions to those rules (and so on) resulting in complicated systems with poor performance.

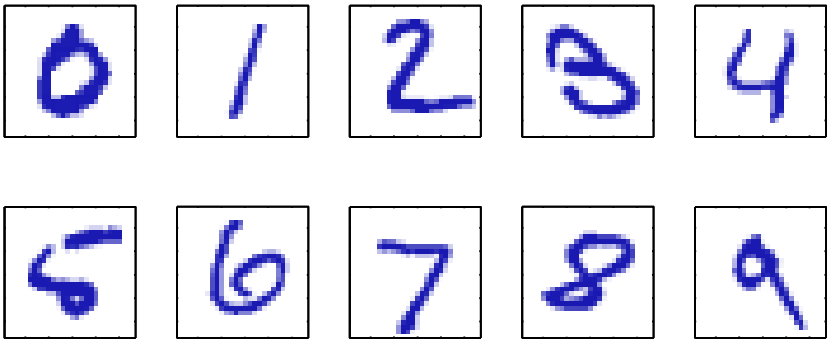


FIGURE 2.1: **Handwritten digits.**
An example of handwritten digits from zip codes in the US, taken from [92].

Much better solutions to this problem is obtained by approaching it from a machine learning perspective, where a set of T digits $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$, referred to as the *training set*, is used for adjusting an adaptive model's parameters. The corresponding category of each digit in the training set, referred to as the *target value* t for each digit $\mathbf{x}$, is known beforehand and form part of the input to the algorithm.

The execution of the algorithm can then be represented by the function $y(\mathbf{x})$, where the digit $\mathbf{x}$ is fed as an input and the algorithm outputs a value y (the identity of the digit) encoded in a way similar to that of the target values. The function's form is determined in the training phase (also called the *learning* phase) using the training data, whereafter it can be used to determine the categories of a set of new digits, called a *test set*. The ability of an algorithm to correctly classify new instances different from those in the training set is referred to as *generalisation*, one of the central goals in machine learning. An important concept to note here is that of *overfitting*. Overfitting can occur when an algorithm is subjected to the learning phase for a too-extensive period of time or when the training set is too small. In cases like this, a learning algorithm often adjusts to certain arbitrary features in the data having no real relation to the problem's target function. During overfitting, an algorithm's performance on the training set will increase while its performance on the test set will decrease.

Usually, the original input features are *pre-processed* in order to express data instances in a new variable space where it would be simpler for the machine learning algorithm to solve the problem. As an example, the images in the above problem have been scaled and translated in order for all digits to be of the same, fixed size - the variability in each category of digits is therefore reduced, making it easier for the algorithm to distinguish correctly between different classes. This process is also sometimes referred to as *feature extraction*, and must be applied to the test set in the same way as it is applied to the training set.

Feature extraction is also carried out for the purpose of speeding up computational processes. In the case of real-time facial recognition, for example, a large number of pixels must be processed by the computer every second. Instead of feeding all these pixels directly into a complex machine learning algorithm, which might be computationally infeasible, one can extract features that can be calculated swiftly and that still preserves the necessary discriminatory information that would be useful for the algorithm when trying to distinguish between faces. These extracted features, of a number less than the original number of pixels in an image, are then the algorithm's new inputs. This type of feature extraction is an example of dimensionality reduction - each face (or data *instance*) is now represented by a few features instead of all its pixels. One must take care to not discard valuable information during the feature extraction phase, as this will lead to an overall decrease in performance of the specific algorithm.

Another motivation for characterising instances with smaller rather than larger numbers of features is the range of phenomena referred to as the *curse of dimensionality*, a term coined by Richard Bellman in 1961 [94]. In the field of machine learning, as explained in the book by Hastie et al. [95], the curse refers to the complexity of learning functions that can grow exponentially with the input feature space dimension, such that in order to estimate these higher-dimensional functions with an accuracy similar to that of the functions in lower dimensions, the training set has to grow exponentially in size as well. Another way of seeing this is to note that some learning algorithms learning from a finite number of training instances with a higher-dimensional feature space (where every feature can take on one of many possible values) requires a vast amount of training data in order to ensure that several instances exist for every combination of possible feature values. If the size of the training set is fixed, the performance of an algorithm decreases as the dimensionality of the feature space increases. This is referred to as the Hughes effect [96]. Other examples of the curse of dimensionality also exist - see section 2.2.2 for a case of the curse that nearest neighbours learning algorithms are usually concerned with.

2.1.2 Learning Categories

Machine learning applications can be grouped into three broad learning categories. Firstly, problems where the training set consists of the input feature vectors $\mathbf{x}$ of the data instances along with

their respective target values t (that need to be predicted) are referred to as *supervised learning*. Supervised learning problems like the digit example used above, where the goal is to map each input $\mathbf{x}$ to one of a number of discrete values (or categories) are known as *classification*, whereas supervised problems in which the input vector is mapped to a real-valued output are known as *regression* problems. An example of a regression problem would be the prediction of property prices in a given city where the size of the property and the distance from the centre of town is given as inputs to the algorithm. Seeing as this thesis is concerned with the problem of classification (as will be explained in Chapter 3), it is again briefly addressed in section 2.1.2.1.

Secondly, applications where the training set consists of the feature vectors $\mathbf{x}$ without their respective target values are referred to as *unsupervised learning* problems. In this category, algorithms can fall into a further one of three broad groups. *Clustering* refers to the finding of similar groups of data instances, for example the finding of tightly-knit communities within vast numbers of people using online social networks. *Density estimation* is the process by which the distribution of the data in the input space is determined, a simple example being the use of a histogram to estimate a continuous variable's probability distribution. *Visualisation* refers to the projection of data from a higher-dimensional to a two- or three-dimensional space.

Lastly, *reinforcement learning* [97] refers to problems trying to find suitable actions in given situations with the goal of maximising a reward over a certain period of time - here, the algorithm does not take optimal output examples during training, but instead has to discover them through trail and error. Reinforcement learning is applied to problems such as robot control, self-driving cars and games such as backgammon, chess and checkers, to name but a few.

2.1.2.1 Supervised Learning: Classification

As already mentioned above, classification problems have as their goal the assignment of a data instance $\mathbf{x}$ to one of M discrete classes $\mathcal{C}_m$, with $m = 1, \ldots, M$. In most classification problems, classes are mutually exclusive, making it impossible to assign an input instance to more than one class. In these cases, we say that the input space is divided into areas called *decision regions*, the boundaries of which are referred to as *decision surfaces*. If the classes in a data set can be perfectly split apart from one another by using only linear decision surfaces, the data is *linearly separable*. Classification problems can be further separated into two categories - in *binary classification* an instance is assigned to one of only two possible classes, while *multiclass classification* involves the mapping of an instance to one of several possible classes.

For this project, we are concerned with the problem of binary classification, as will be explained in Chapter 3. Many existing learning algorithms addresses this task. In the next section we discuss the naïve Bayes classifier, the k-nearest neighbours algorithm, the support vector machine and Artificial Neural Networks, as these were the algorithms selected for the purpose of this project.

2.2 Machine Learning Algorithms

This section discusses the naïve Bayes classifier (section 2.2.1), the k-nearest neighbours algorithm (section 2.2.2), support vector machines (section 2.2.3) and Artificial Neural Networks (section 2.2.4) for the purpose of machine learning classification. It should be noted that the sections describing these algorithms supply the standard theoretical background in order to provide a basis for understanding later chapters more easily. Discussions and explanations of the methods are kept general, with the specific implementation of each of the algorithms and their deviations from the standard theoretical models discussed in Chapter 5.

2.2.1 Naïve Bayes

Bayesian reasoning is built on the supposition that quantities of interest are controlled by probability distributions, and that taking observed data into consideration while working with these probabilities can result in optimal decisions being made. Its importance to machine learning is based on the fact that it can be used for providing quantitative approaches to weighing the evidence for different hypothesis. It provides the framework for algorithms directly calculating and manipulating probabilities.

This section will discuss the naïve Bayes classifier, a highly practical Bayesian classifier, starting first with an introduction to Bayes theorem.

2.2.1.1 Bayes Theorem: A Machine Learning Perspective

In machine learning we often want to determine the most suited, or the *most probable*, hypothesis (outcome) h from some hypothesis space $H = \{h_1, h_2, ..., h_M\}$, given observed training data D as well as any knowledge concerning the prior probabilities of the different hypotheses.

Bayes theorem gives us a way of calculating how probable different hypotheses are. By using the prior probability of a hypothesis, the observed training data and probabilities of observing certain data given the hypothesis, we can determine the hypothesis's probability.

In order to express Bayes theorem mathematically, some notation should be introduced. $P(h)$ denotes the *prior probability* of hypothesis h – that is, it is the probability of hypothesis h being the correct one, before any training data have been observed. In the same way, $P(D)$ denotes the prior probability of observing the training data, given no information as to which hypothesis is the correct one, while $P(D|h)$ is the probability that D, the training data, will be observed given that hypothesis h holds. Our main interest here, however, is in finding the *posterior probability* of a certain hypothesis h, denoted by $P(h|D)$, as this is a reflection of our confidence that a specific

hypothesis holds *after* seeing the training data D. Bayes theorem gives us a method of calculating $P(h|D)$ using the prior probabilities $P(h)$ and $P(D)$, as well as $P(D|h)$, and is defined in Eq. 2.1.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \tag{2.1}$$

In many classification machine learning problems, the learning algorithm considers a set of hypotheses (classes) H and tries to find the hypothesis $h \in H$ that is most probable to be correct, given the training data D. Such a most probable hypothesis can be referred to as a "maximum a posteriori" (MAP) hypothesis, and can be determined by calculating each hypothesis's posterior probability with the use of Bayes theorem. Mathematically, h_{MAP} is a MAP hypothesis if

$$\begin{aligned} h_{MAP} &\equiv \underset{h \in H}{\operatorname{argmax}}\, P(h|D) \\ &= \underset{h \in H}{\operatorname{argmax}}\, \frac{P(D|h)P(h)}{P(D)} \end{aligned} \tag{2.2}$$

Here, argmax is short for the "argument of the maximum", an operator returning the points of the argument that maximises the given function. For example,

$$\underset{x}{\operatorname{argmax}}\, f(x) \tag{2.3}$$

are those points x that will maximise the function $f(x)$. The operator argmin ("argument of the minimum") can be defined analogously. Noting next that the term $P(D)$ in Eq. 2.2 is constant and not dependent on h, we can drop it and simplify the equation to

$$h_{MAP} = \underset{h \in H}{\operatorname{argmax}}\, P(D|h)P(h). \tag{2.4}$$

Sometimes, for example when we have no prior information about any of the candidate hypotheses in H, it is appropriate to assume that all of them have an equal a priory probability of being correct. That is, $P(h_j) = P(h_i)$ for all h_j and h_i in H. In such cases Eq. 2.4 can be further simplified by dropping the term $P(h)$ so that only $P(D|h)$ needs to be considered for finding the MAP hypothesis. $P(D|h)$ is also referred to as the *likelihood* of D given h, and the hypothesis maximising $P(D|h)$ is known as the *maximum likelihood* (ML) hypothesis, h_{ML}. Taking all this into account, Eq. 2.4 can now be re-written as follows:

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}}\, P(D|h) \tag{2.5}$$

2.2.1.2 The Naïve Bayes Classifier

The naïve Bayes classifier is a Bayesian learning algorithm that can be applied to learning problems where each data instance is characterised by a feature vector $\mathbf{x}$, and where $y(\mathbf{x})$, the target function, can be assigned any one of M discrete target values $\mathcal{C}_m$, where $m = 1, \ldots, M$ (this can be seen as equivalent to the hypothesis space H mentioned in section 2.2.1.1).

Training instances for each of the possible target values of the target function is provided to train the classifier, after which a new instance, described by a feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)$, is presented. The classifier is then asked to classify the new instance.

Using the Bayesian approach, this classification is done by finding the target value with the highest probability of being correct, $\mathcal{C}_{MAP}$, given the feature values $(x_1, x_2, \ldots, x_N)$ describing the instance. This is expressed mathematically as follows:

$$\mathcal{C}_{MAP} = \underset{\mathcal{C}_m}{\operatorname{argmax}}\, P(\mathcal{C}_m | x_1, x_2, \ldots, x_N) \tag{2.6}$$

Using Bayes theorem, defined in Eq. 2.1, and applying the same simplification that was used to get to Eq. 2.4, we can rewrite Eq. 2.6 as

$$\begin{aligned} \mathcal{C}_{MAP} &= \underset{\mathcal{C}_m}{\operatorname{argmax}}\, \frac{P(x_1, x_2, \ldots, x_N | \mathcal{C}_m) P(\mathcal{C}_m)}{P(x_1, x_2, \ldots, x_N)} \\ &= \underset{\mathcal{C}_m}{\operatorname{argmax}}\, P(x_1, x_2, \ldots, x_N | \mathcal{C}_m) P(\mathcal{C}_m) \end{aligned} \tag{2.7}$$

The two terms in the last line of Eq. 2.7 can now be estimated with the help of the training data. The $P(\mathcal{C}_m)$ terms can easily be estimated merely by determining the frequency of occurrence of each target value $\mathcal{C}_m$ in the training data. It is, however, not feasible to determine the large number of $P(x_1, x_2, \ldots, x_N | \mathcal{C}_m)$ terms[1] in this way if we do not have a very large training data set, as reliable estimates can only be made if every possible instance in the instance space is observed many times in the training set.

If we now assume conditional independence of the feature values given $\mathcal{C}_m$, the probability of observing the feature vector $(x_1, x_2, \ldots, x_N)$ given the target value of the instance is merely the product of the individual features' probabilities, as follows:

$$P(x_1, x_2, \ldots, x_N | \mathcal{C}_m) = \prod_i P(x_i | \mathcal{C}_m) \tag{2.8}$$

[1] The number of possible classes multiplied by the number of possible data instances.

We can substitute Eq. 2.8 into Eq. 2.7 to obtain the naïve Bayes classifier's approach to classification, shown in Eq. 2.9.

$$\mathcal{C}_{NB} = \underset{\mathcal{C}_m}{\operatorname{argmax}}\, P(\mathcal{C}_m) \prod_i P(x_i|\mathcal{C}_m) \tag{2.9}$$

Here, $\mathcal{C}_{NB}$ is the class output of the learning algorithm. Now, instead of having to calculate a large number of $P(x_1, x_2, \ldots, x_N|\mathcal{C}_m)$ terms as before, we have to estimate the $P(x_i|\mathcal{C}_m)$ terms. Seeing as the number of these terms is merely the number of classes multiplied by the number of feature values, we are now left to estimate a much smaller number of terms than before.

In summary, the training phase of the naïve Bayes classifier consists of estimating the $P(\mathcal{C}_m)$ and $P(x_i|\mathcal{C}_m)$ terms - this is done by counting their respective frequencies in the training data. The exact way in which this is done varies in practice - to see how we implemented the estimation of these terms, see section 5.4.2.1. Once the learning algorithm is trained, it can be used for the classification of new data instances by using the approach in Eq. 2.9, where $\mathcal{C}_{NB}$ is the same as the MAP classification if the simplifying assumption of conditional independence discussed earlier is satisfied.

This simplifying assumption is also the one major setback of the naïve Bayes classifier. Feature values (given a class $\mathcal{C}_m$) are rarely conditionally independent in real-life problems, an issue that often leads to the reduced performance of the classifier.

2.2.2 k-Nearest Neighbours

The k-nearest neighbours algorithm is an extremely simple machine learning algorithm. A new unclassified test instance $\mathbf{x}$ is classified by carrying out a majority class vote among the k training instances that are closest to it in feature space. These k training instances are referred to as its k nearest neighbours. [95]

To intuitively illustrate the working of the algorithm, see Fig. 2.2, a visualisation of the classification of a test instance $\mathbf{x}_0$ by a 5-nearest neighbours ($k = 5$) algorithm. Here, the training instances are two-dimensional data points (each instance is represented by $N = 2$ features) with either a positive or a negative class-value, denoted by "+" and "-", respectively. In this case $\mathbf{x}_0$ will be classified as "+", as three of its five nearest neighbours (encircled in the figure) is of the "+" class while only two of its nearest neighbours are of the "-" class - a majority class vote is therefore in favour of the positive class.

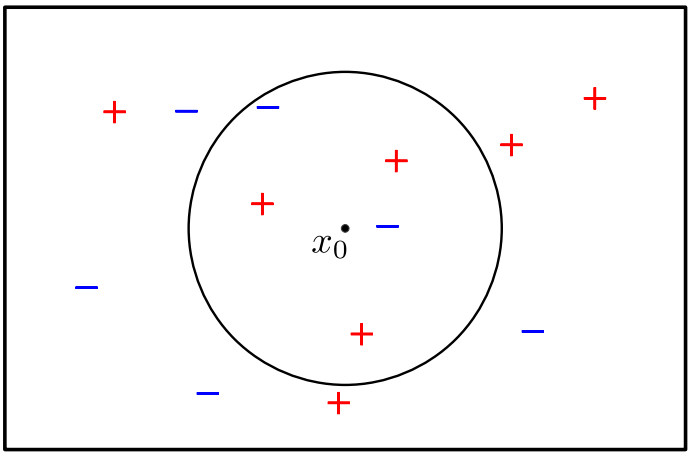


FIGURE 2.2: **A k-nearest neighbours classification example.**
An example classification of a test instance $\mathbf{x}_0$ by a 5-nearest neighbours ($k = 5$) algorithm into either a positive or a negative class, denoted by "+" and "-" respectively. Because the majority of the five closest training instances (encircled) are positive, $\mathbf{x}_0$ will be classified as belonging to the positive class.

In order to determine the k nearest neighbours of a particular test instance, one has to select the k training instances that are closest in distance to it in feature space (as mentioned above). If the test instance $\mathbf{x}$ has a feature vector

$$(x_1, x_2, ..., x_N) \tag{2.10}$$

where x_n is the n'th feature of $\mathbf{x}$, then the Euclidean distance between $\mathbf{x}$ and a training instance $\mathbf{x}_j$ is denoted as $d(\mathbf{x}, \mathbf{x}_j)$, given as

$$d(\mathbf{x}, \mathbf{x}_j) = \sqrt{\sum_{n=1}^{N} (x_n - x_{j,n})^2}. \tag{2.11}$$

To mathematically express the intuitive picture in Fig. 2.2, we consider only discrete-valued target functions (for classification, as mentioned before) denoted by $y(\mathbf{x})$, where y can be assigned any one of M discrete target values $\mathcal{C}_m$, where $m = 1, \ldots, M$. The k-nearest neighbours algorithm for the target function of a new test instance $\mathbf{x}$ can then be expressed as

$$y(\mathbf{x}) = \underset{\mathcal{C}_m}{\operatorname{argmax}} \sum_{r=1}^{k} \delta(\mathcal{C}_m, t_r) \tag{2.12}$$

where t_r is the class value (target value) of the r'th nearest neighbour and $\delta(a, b) = 1$ if $a = b$ and $\delta(a, b) = 0$ otherwise. It can therefore be seen that $y(\mathbf{x})$ is simply the most common class value of $\mathbf{x}$'s k-nearest neighbours.

Considering Fig. 2.2 one last time, it can be seen that in the case of using a 2-nearest neighbours algorithm ($k = 2$), there would be an equal count of "+" and "-" classes among the neighbours of $\mathbf{x}_0$. There are different ways of solving this problem. One can, for example, increase or decrease k so that a majority vote will be able to yield one class as the winner. A good way of mostly avoiding such a problem, however, is to make use of a common extension to the k-nearest neighbours algorithm: weighting the contribution of each of the k-nearest neighbours with regard to their distance from the test instance. Introducing such weights, Eq. 2.12 then becomes

$$y(\mathbf{x}) = \underset{\mathcal{C}_m}{\operatorname{argmax}} \sum_{r=1}^{k} w_r \delta(\mathcal{C}_m, t_r), \tag{2.13}$$

where w_r is the weight of the r'th nearest neighbour. One possible example of such a weight can be seen in Eq. 2.14, where the importance of nearest neighbours will be scaled by the inverse square of the distance between the particular neighbour $\mathbf{x}_r$ and the test instance $\mathbf{x}$.

$$w_r = \frac{1}{d(\mathbf{x}, \mathbf{x}_r)^2}. \tag{2.14}$$

2.2.2.1 Remarks on the Algorithm

One possible disadvantage of the k-nearest neighbours method is concerned with the fact that all the features of instances are used when calculating the Euclidean distance between data points. If it so happens that only a small set of the features contain discriminatory information while the remaining majority of features are irrelevant, the distance between instances end up being more influenced by the irrelevant features than the useful ones. This problem, arising with the presence of a large number of irrelevant features, is another case referred to as the *curse of dimensionality* (see section 2.1.1). Nearest neighbours methods are particularly sensitive to this specific example of the curse. This can be partially overcome, however, by weighting each feature differently when calculating distances between instances, or by removing the irrelevant features from the data entirely.

Another issue with the k-nearest neighbours algorithm is that, because the k closest training instances to a test instance is only determined upon its presentation for classification, a significant amount of computation might be necessary for each new classification. Many approaches for efficient memory indexing of the training instances have been developed in order to more easily identify the relevant nearest neighbours at the cost of using more memory. An example of such an approach is the *kd*-tree [98, 99], with training instances being stored as leaves in a tree. Instances close to each other are then stored on the same branch or node, making it less computationally intensive to locate the neighbours of a particular instance.

An important difference between k-nearest neighbours and most other machine learning techniques is the fact that, for each unclassified test instance encountered, k-nearest neighbours creates a unique approximation to the target function, whereas most other methods construct approximations that are applied over the whole instance space that is not localised to the neighbouring area of a new test instance. This is a big advantage in cases where the global target function is complex. [93]

Because the k-nearest neighbours method is highly unstructured, it is not necessarily of great value when trying to understand the relationships between the feature values and the classification outcome. However, when used as a black-box approach, they often prove to be some of the best-performing algorithms in real-life machine learning problems. [95]

2.2.3 Support Vector Machine

The support vector machine is an example of a maximum margin classifier, which means that the algorithm's aim is in finding a decision boundary that separates different classes from one another, while also seeking to maximise the separation between them. In the algorithm's simplest implementation, the decision boundary takes the form of a hyperplane[2] in feature space. Its strength, though, lies in its ability to transform the data features into a higher-dimensional space with the help of the *kernel trick* (see the next section and Fig. 2.5) where a hyperplane will be more successful in its task of separating classes as well as possible. This then results in a non-linear decision boundary in the original feature space.

In this section, the support vector machine will be discussed by first looking at the case for linearly separable data, then briefly looking at kernel functions (as these are integral to the algorithm) and then considering the case for non-linearly separable data.

2.2.3.1 Formulation of the Support Vector Machine Algorithm

We start off by looking at a binary classification problem making use of linear models with the following form:

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}) + b \tag{2.15}$$

Here, b is a *bias* parameter determining the decision surface's location, $\mathbf{w}$ is the *weight vector* determining the decision surface's orientation, and $\phi(\mathbf{x})$ is a feature-space transformation. The training set consists of N input feature vectors $\mathbf{x}_1, ..., \mathbf{x}_N$ having class values $t_1, ..., t_N$, respectively,

[2] In N-dimensional Euclidean space, a hyperplane is an $N-1$-dimensional flat subset of the space dividing it into two subspaces.

where $t_n \in \{-1, 1\}$. New data instances $\mathbf{x}$ are then classified into one of these two classes depending on the sign of the target function $y(\mathbf{x})$, meaning that the decision boundary is defined by the relation $y(\mathbf{x}) = 0$.

For now, we'll assume that the instances in the training data are linearly separable. In other words, there is at least one combination of values of $\mathbf{w}$ and b such that Eq. 2.15 satisfies $y(\mathbf{x}_n) > 0$ for all training instances having $t_n = +1$, and $y(\mathbf{x}_n) < 0$ for all training instances having $t_n = -1$. We therefore have that $t_n y(\mathbf{x}_n) > 0$ for all instances in the training set.

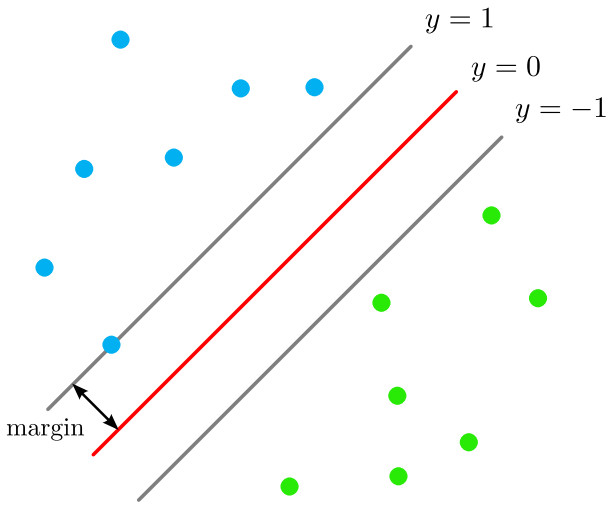


FIGURE 2.3: **The concept of a margin.**
This figure shows a linearly separable training data set with instances belonging to one of two possible classes (shown by the blue and green dots, respectively). The margin is the smallest perpendicular distance between any of the training instances and the decision boundary (shown by the red line at $y = 0$).

It may of course be that there are many solutions for $\mathbf{w}$ and b that results in a complete separation of classes for the training data set. In that case, we choose the solution that will result in the smallest classification error when the algorithm is applied to new instances in a validation set. The support vector machine tackles this problem by using the concept of a *margin*, defined as the smallest perpendicular distance between any of the training instances and the decision boundary, as illustrated in Fig. 2.3.

For a support vector machine, minimising the classification error of new test instances means that the decision boundary is chosen such that the margin is maximised (based on the training data) - from there the saying that it is a maximum margin classifier.

To implement a support vector machine it is necessary to determine the parameters $\mathbf{w}$ and b in Eq. 2.15. It can be shown, and is fully derived in Appendix A.1, that this optimisation problem can be reduced to solving

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2, \tag{2.16}$$

subject to the constraint

$$t_n\left(\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)+b\right) \geq 1, \qquad n = 1, ..., N \tag{2.17}$$

where $n = 1, ..., N$. Because we are faced with having to solve a constrained optimisation problem, we use the Lagrange multipliers[3] $a_n \geq 0$, one for each constraint mentioned above, resulting in the Lagrangian function

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{n=1}^{N} a_n \left\{t_n\left(\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)+b\right) - 1\right\}, \tag{2.18}$$

where $\mathbf{a}$ is the vector of Lagrange multipliers. This complex problem is recast in a more tractable form and solved in Appendix A.2, where the values of $\mathbf{a}$, $\mathbf{w}$ and b are determined.

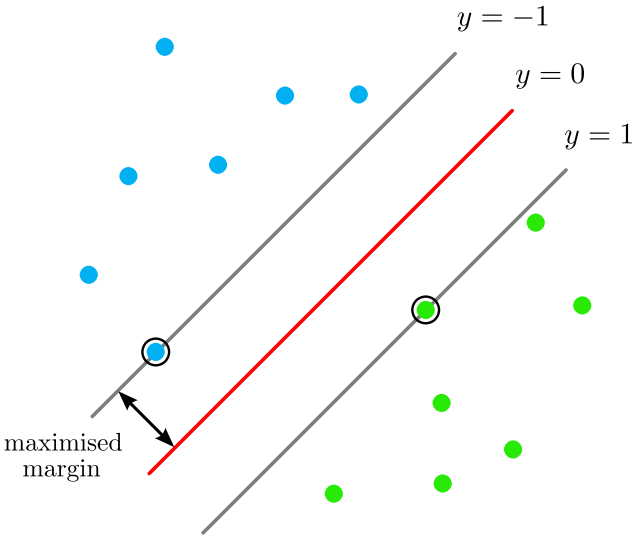


FIGURE 2.4: **Maximised margins and support vectors.**
When maximising the margin, support vectors (the circled instances) determine the location of the decision surface.

These solutions can be understood intuitively as follows. For every training instance $\mathbf{x}_n$, either $a_n = 0$ or $t_n y(\mathbf{x}_n) = 1$. Instances for which $a_n = 0$ will have no influence over the prediction for a new test instance. The remaining training instances, for which $t_n y(\mathbf{x}) = 1$, are referred to as *support vectors* and lie on the hyperplanes in feature space corresponding to the maximum margins,

[3]For information on Lagrange multipliers, see Appendix E in Bishop's book [92].

as is shown in Fig. 2.4. It is therefore important to note that, upon completion of the training phase, only the instances corresponding to the support vectors are retained for the classification of new test instances.

2.2.3.2 Kernel Functions

For learning models using a fixed feature space transformation $\boldsymbol{\phi}(\mathbf{x})$, where $\boldsymbol{\phi}(\mathbf{x})$ is a vector and $\mathbf{x}$ is its input feature vector, the kernel function is defined as

$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}') \tag{2.19}$$

and is a symmetric function of its inputs, such that

$$k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x}) \tag{2.20}$$

Many linear parametric regression and classification models can be reformulated into a *dual representation*, based on the use of a kernel function evaluated using the training instances. This approach was pioneered by Aizerman et al. in 1964 [100] with applications in pattern recognition, and was rediscovered years later in 1992 by Boser et al. [101], who applied it in a machine learning context that led to the development of the support vector machine algorithm.

Formulating a kernel as an inner product in feature space allows for the use of the *kernel trick*, saying that if an algorithm has input instances $\mathbf{x}$ entering only as scalar products, the product can be replaced by a kernel to obtain a dual algorithm equivalent to the original. In this way we end up with a dual algorithm expressed entirely with kernels, and can therefore avoid explicitly working with $\boldsymbol{\phi}(\mathbf{x})$. This allows us the implicit use of feature spaces having higher or infinite dimensionality. To see an example of the application of the kernel trick, see Fig. 2.5.

There are two ways in which a valid kernel function can be constructed. The first is to define a feature mapping $\boldsymbol{\phi}(\mathbf{x})$ and to then construct a kernel function from that, while the second approach is to construct it directly. In the second case it is necessary to check the validity of the chosen function as a kernel by ensuring that it is a scalar product in some feature space.

However, we need a simpler way of determining whether a kernel is valid without having to determine $\boldsymbol{\phi}(\mathbf{x})$ explicitly. For this reason we introduce the Gram matrix, denoted by $\mathbf{K}$, a symmetric $N \times N$ matrix having elements

$$K_{nm} = \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) \tag{2.21}$$

For a kernel $k(\mathbf{x}, \mathbf{x}')$ to be valid, a necessary and sufficient condition [102] is for the Gram matrix to be positive semidefinite[4] for all possible sets $\{\mathbf{x}_n\}$. Lastly, new valid kernels can also be constructed by building them using other kernels as building blocks (see the properties in Appendix A.4).

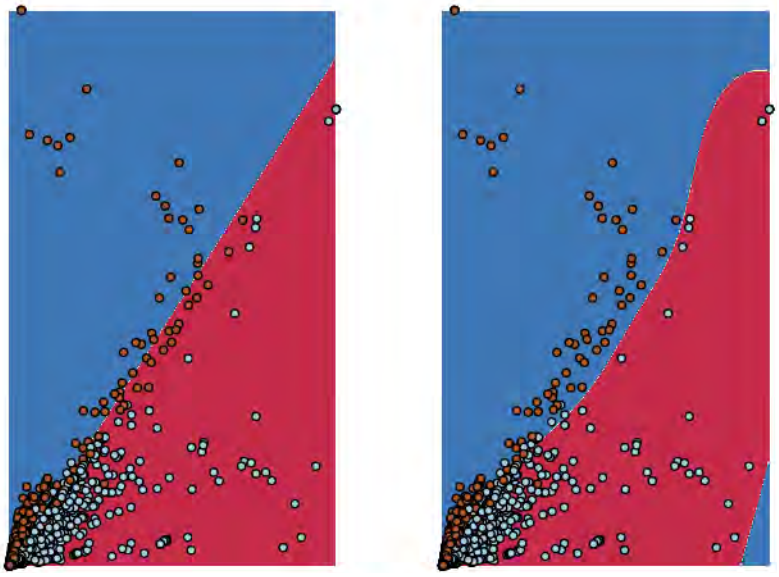


FIGURE 2.5: **The kernel trick.**
Left: Two-class classification with a support vector machine using a linear decision boundary.
Right: Two-class classification with a support vector machine making use of the kernel trick.

2.2.3.3 Linearly Non-Separable Class Distributions

Up to this point, it was assumed that the training set was linearly separable in some feature space $\phi(\mathbf{x})$, meaning that using a support vector machine will lead to a complete separation of the classes found in the training set in the original input feature space $\mathbf{x}$, where the separation will be done by a non-linear decision surface. In real-life applications, on the other hand, class-conditional distributions often overlap, and finding a complete separation of the classes in the training data is often detrimental to the classifier's performance on the test set later on.

In order to prevent this overfitting from taking place, the support vector machine needs to be modified in order to allow for the misclassification of some of the instances in the training set. In other words, we now want to allow training instances to fall on the "other side" of the margin hyperplane. To exercise a measure of control over which instances gets misclassified and which don't, penalties are introduced, implemented through the use of *slack variables* defined as $\xi_n \geq 0$ with $n = 1, \ldots, N$ [103, 104]. Each instance has one slack variable, its value increasing as the distance between the instance and the margin boundary increases. We here take the penalty to be

[4] An $N \times N$ matrix $\mathbf{M}$ is positive semidefinite if $x^*\mathbf{M}x \geq 0$, where the column vector $x \in \mathbb{C}^N$ if $\mathbf{M}$ is complex, or $x \in \mathbb{R}^N$ if $\mathbf{M}$ is real. Note also that x^* is the complex conjugate of x.

a linear function of the aforementioned distance, as this will prove to be most convenient for the optimisation problem we have to solve.

For training instances lying inside or on the correct margin hyperplane, the slack variables are defined as $\xi_n = 0$, while for all other instances they are assigned the value $\xi_n = |t_n - y(\mathbf{x}_n)|$. A training instance falling exactly on the decision surface $y(\mathbf{x}_n) = 0$ will therefore have a penalty of $\xi_n = 1$, while instances having $\xi_n > 1$ lies at a point on the other side of the decision surface, and will be misclassified. The classification constraint given in Eq. 2.17 for the case of separable classes are therefore replaced with

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n, \qquad n = 1, ..., N \tag{2.22}$$

where the penalties are constrained to $\xi_n \geq 0$. Training instances falling on the margin hyperplane or on the correct side of the margin will have slack variables with the value $\xi_n = 0$ and will be classified correctly, as will instances falling between the margin hyperplane and the decision surface on the correct side having slack variables of the value $0 < \xi_n \leq 1$. Instances falling on the decision boundary's wrong side will have slack variables $\xi_n > 1$, however, and will be misclassified. This is illustrated in Fig. 2.6. This relaxed margin constraint, resulting in what we now refer to as a *soft margin*, therefore allows for the occasional misclassification of training instances. While this approach allows class distributions to overlap, it remains sensitive to training data outliers due to the fact that the misclassification penalty is linearly increasing with ξ.

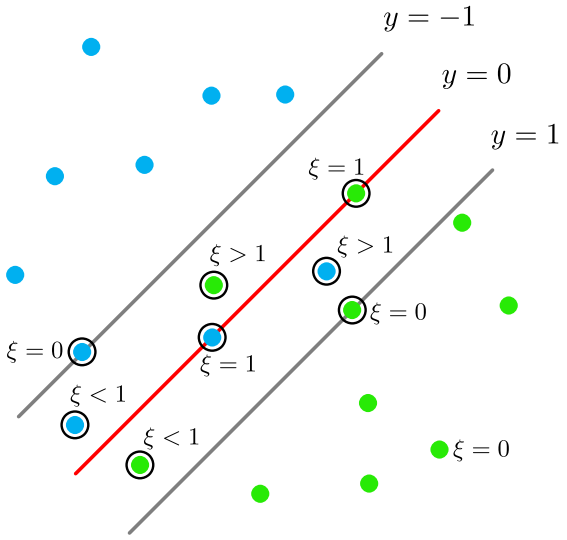


FIGURE 2.6: **Slack variables.**
This figure shows the slack variables, ξ_n, for different support vectors (circled data instances).

Our aim, now, is again to maximise the margin, this time while penalising instances falling inside the margin or on the margin hyperplane itself. This is done by minimising

$$C\sum_{n=1}^{N}\xi_n + \frac{1}{2}||\mathbf{w}||^2 \tag{2.23}$$

with respect to $\mathbf{w}$, b and $\{\xi_n\}$. As misclassified instances have $\xi_n > 1$, $\sum_N \xi_n$ forms the upper bound to the number of misclassifications. C, where $C > 0$, therefore serves as a regularisation parameter that manages the trade-off between the minimisation of errors (the penalties) and the control of the model's complexity (the margin). A low C results in a smooth decision boundary and a high C results in greater classification accuracy. To recover the algorithm used for the case of separable classes, one merely takes the limit $C \to \infty$. We highlight here that the parameter C is user-defined, as is explained later when our particular support vector machine implementation is discussed (section 5.4.3).

We now minimise Eq. 2.23 while taking the constraints in Eq. 2.22 as well as $\xi_n > 0$ into account. Introducing $\{a_n \geq 0\}$ and $\{\mu_n \geq 0\}$ as Lagrange multipliers, we therefore solve the following Lagrangian function:

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N}a_n\{t_n y(\mathbf{x}_n) - 1 + \xi_n\} - \sum_{n=1}^{N}\mu_n\xi_n \tag{2.24}$$

As before, the solutions for $\mathbf{w}$, b and $\mathbf{a}$ are given in Appendix A.3.

2.2.3.4 Obtaining Probabilistic Outputs

Support vector machines are decision machines, and therefore do not output posterior probabilities for the target values of new test instances. In order to obtain such probabilistic outputs, Platt scaling [105] can be employed - this involves the fitting of a logistic sigmoid, an S-shaped curve defined by

$$\sigma(r) = \frac{1}{1 + e^{-r}} \tag{2.25}$$

to the outputs of a support vector machine that has already been trained. It is then assumed that the conditional probability we are looking for is as follows:

$$p(t = 1|\mathbf{x}) = \sigma(Ay(\mathbf{x}) + B) \tag{2.26}$$

Here, $y(\mathbf{x})$ is defined by Eq. 2.15. A and B can be determined through a maximum likelihood method optimising on a training set that is independent of the training data used for the initial training of the support vector machine so as to avoid over-fitting. In the case where one training set is used for both purposes, an additional cross-validation can be used to avoid over-fitting.

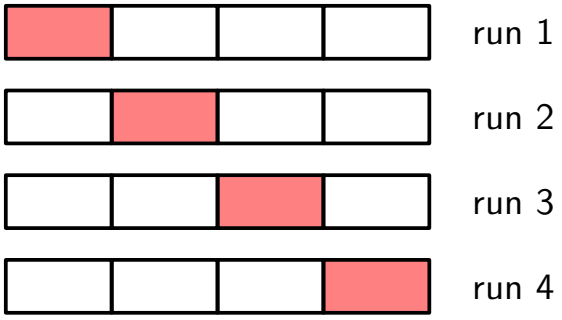


FIGURE 2.7: **N-fold cross-validation.**
This is an illustration of N-fold cross-validation on a set of data, where $N = 4$. For each run, 3 of the 4 equally-sized groups are used for model training while the withheld group (in pink) is used for model validation. This figure was taken from [92].

Cross-validation is a technique used for determining how results obtained from a predictive model (in the case of machine learning) will generalise to another, independent set of data. The method uses a fraction $(T-1)/T$ of the training data for training while systematically employing the whole training set for performance assessment. Consider the case of 4-fold cross-validation ($T = 4$) where the training set has been split into four equally-sized groups, as illustrated in Fig. 2.7. $T - 1$ of these groups are then used for the training of a model, while the remaining group is used for the evaluation of the trained model. This procedure is repeated T times, once for each of the possible choices for the withheld group (indicated with pink in the figure), after which the T performance results are averaged.

2.2.4 Artificial Neural Network

Artificial neural networks have their origin in the study of mathematical representations of biological neural networks [4, 22, 24]. Historically, these algorithms have since been researched by two groups of scientists - the one group, as alluded to above, using neural networks to study learning processes in biological entities; the other group studying them in the pursuit of developing better machine learning methods [93]. For our purposes a biological framework and the corresponding constraints it imposes is of course unnecessary, and this section will therefore concentrate on introducing neural networks in the proper machine learning context by discussing a simple class of networks, called feed-forward network functions, that have shown themselves to be of great value.

Neural networks are nonlinear modelling tools able to approximate target functions that are real-, discrete- or vector-valued. They consist of a system of interconnected nodes, of which each node processes received information before passing it onwards to other nodes using weighted links. These networks can be trained to learn mappings from inputs to outputs, after which it can then use the trained model to make predictions regarding new test data instances. [88]

In this section, we discuss artificial neural networks as supervised machine learning algorithms used for the purpose of binary classification. The perceptron, one of the basic units of neural networks, will be discussed first, followed by a description of feed-forward neural networks. Autoencoders, a type of feed-forward network, will be discussed next, after which there will be looked at the training of neural networks. Lastly, we discuss the method of error backpropagation.

2.2.4.1 The Perceptron

The perceptron, developed by Frank Rosenblatt [24], forms an integral part of the history of artificial intelligence and machine learning in the context of neural networks. The perceptron is a binary classifier that maps an input N-dimensional feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ to either class $\mathcal{C}_1$ or $\mathcal{C}_2$, denoted by the target values $t = 1$ and $t = -1$, respectively. [92]

The perceptron achieves this by, given the input $\mathbf{x}$, calculating the target function $y(\mathbf{x})$ as

$$y(\mathbf{x}) = \begin{cases} +1 & \text{if} \quad \sum_{i=1}^{N} w_i x_i + w_0 \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.27}$$

where w_i, with $i = 1, \ldots, N$, are real-valued *weights* determining the contribution that each one of the feature inputs x_i has to the output of the perceptron, and where w_0 denotes a *bias* parameter [93]. To simplify notation and for further illustration, see Fig. 2.8, where the feature inputs, weights and bias parameter can be seen on the left, with their sum a expressed as

$$a = \sum_{i=1}^{N} w_i x_i + w_0 \tag{2.28}$$

The function $h(\cdot)$ is known as the *activation* function, and in the case of the perceptron algorithm is the signum step function. Mathematically, the target function is therefore given by

$$y(\mathbf{x}) \;=\; h(a) \;=\; \operatorname{sgn}(a) \;=\; \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases} \tag{2.29}$$

where a is as shown in Eq. 2.28.

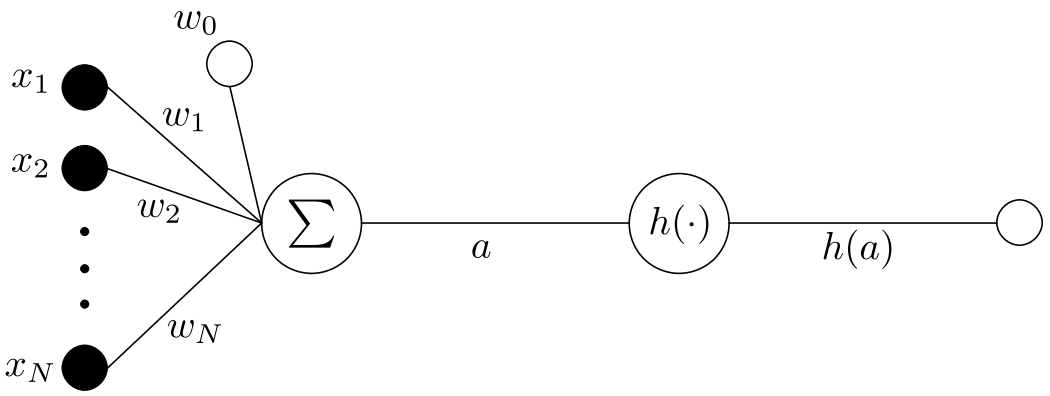


FIGURE 2.8: **The perceptron.**
An illustration of the perceptron classifier, where x_i and w_i, with $i = 1, \ldots, N$, denotes the feature inputs and their weights, respectively, and where w_0 denotes the bias parameter. The weighted sum of the feature inputs is denoted by a (Eq. 2.28), while h is referred to as the activation function (see Eq. 2.29).

The perceptron classifier is trained using a set of training instances $\{\mathbf{x}_n\}$ to choose values for the weights w_i. Once training is completed and a reliable mapping between the training inputs and their target values is established, the model can be used for the classification of new test input instances. Because we are primarily interested in feed-forward networks, further details regarding the training of perceptrons is not necessary.

2.2.4.2 Feed-Forward Neural Networks

Generally, neural networks can have arbitrary structures, but because a great number of learning applications can be implemented using only feed-forward neural networks [88], it is this type of network that will be discussed in this section. Feed-forward neural networks are the simplest networks and have directed structures - information moves from an input to an output nodal layer via a number of "hidden" node-layers in between [88]. The feed-forward network is often referred to as a *multilayer perceptron*, even though the simple nodes used in it are not strictly speaking perceptrons (discussed in the above section) in that their activation functions differ. This section discusses the feed-forward neural network algorithm.

For simplicity we consider a 3-layer neural network, shown in Fig. 2.9, consisting of an input layer with N nodes, a hidden layer with M nodes and an output layer with K nodes. Note that the presence of x_0 and z_0 in the above figure will be made clear in Eq. 2.36, and can be ignored for now. To determine the hidden layer node outputs, denoted by z_j, we start by constructing M linear combinations (one for each of the hidden nodes) of the input feature values $x_1, \ldots, x_N$ of an instance $\mathbf{x}$. This can be expressed in a similar fashion as Eq. 2.28, as

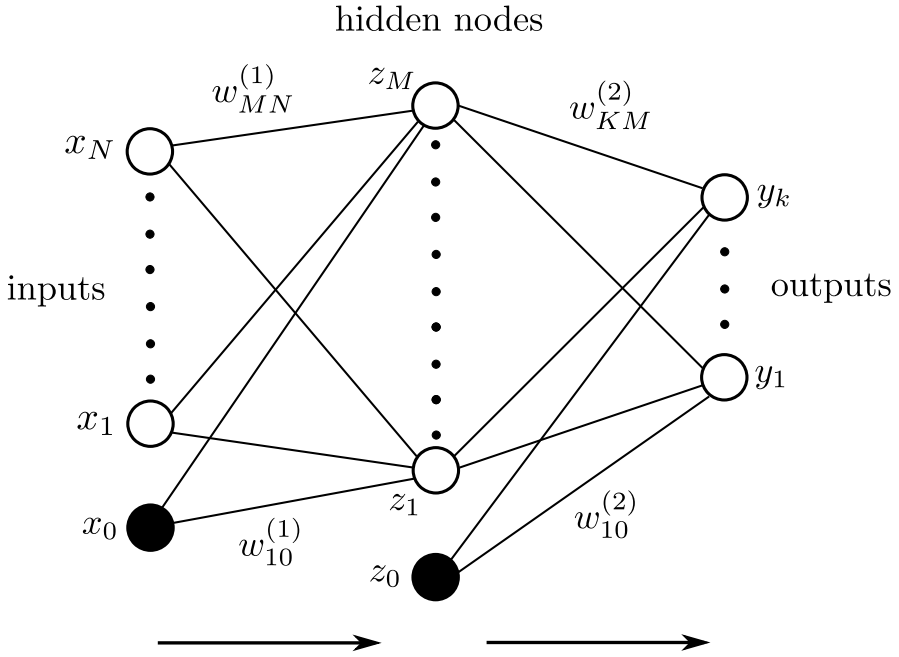


FIGURE 2.9: **A 3-layer feed-forward neural network.**
A 3-layer neural network corresponding to Eq. 2.36 showing an input, hidden and output layer, with variables represented by nodes, weights depicted with links, and bias parameters represented by links from the extra parameters x_0 and z_0, where $x_0 = 1$ and $z_0 = 1$. Information flows in the direction shown by the arrows.

$$a_j = \sum_{i=1}^{N} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{2.30}$$

with $j = 1, \ldots, M$ and where the superscript (1) means that the parameters are associated with the first network layer. Here, in a similar fashion as in the case with the perceptron, the $w_{ji}^{(1)}$ parameters will be referred to as *weights* while the $w_{j0}^{(1)}$ parameters will be called *biases*. The a_j quantities will be referred to as *activations*, and will be transformed by a nonlinear, differentiable activation function $g(\cdot)^{(1)}$ to give the outputs z_j of the hidden layer nodes. This is as follows:

$$z_j = g^{(1)}(a_j) \tag{2.31}$$

The nonlinear activation functions used for feed-forward neural networks are usually taken to be sigmoidal, the logistic sigmoid function (on the left in Eq. 2.32) or 'tanh' function (on the right in Eq. 2.32) being popular choices. They differ from the perceptron's activation function in that they are differentiable.

$$g(x) = \frac{1}{1+e^{-x}} \qquad g(x) = \tanh(x) \tag{2.32}$$

To determine the outputs of the nodes in the third layer of the neural network, we next construct K linear combinations of the z_j values determined above to obtain the output node activations a_k, as

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{2.33}$$

with $k = 1, \ldots, K$, and where the superscript (2) indicates that the parameters correspond to the second layer of the network. Again, the $w_{kj}^{(2)}$ parameters are weights and the $w_{k0}^{(2)}$ parameters are biases. The output node activations are then transformed with an activation function $g(\cdot)^{(2)}$ to yield the outputs of the network, y_k, as follows:

$$y_k = g^{(2)}(a_k) \tag{2.34}$$

Combining these stages, we can write down the overall target function $y_k(\mathbf{x}, \mathbf{w})$ for the 3-layer neural network as

$$y_k(\mathbf{x}, \mathbf{w}) = g^{(2)}\left(\sum_{j=1}^{M} w_{kj}^{(2)} g^{(1)}\left(\sum_{i=1}^{N} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{2.35}$$

with all the parameters for the weights and biases grouped together as $\mathbf{w}$. The neural network algorithm is therefore a nonlinear function mapping the inputs $\mathbf{x} = (x_i, \ldots, x_N)$ to the outputs $\{y_k\}$ using a vector of adjustable parameters $\mathbf{w}$, where network training is carried out to determine the values of $\mathbf{w}$. To simplify notation we can introduce the extra inputs $x_0 = 1$ and $z_0 = 1$ so that the bias parameters can effectively form a part of the set of weights. Eq. 2.35 therefore becomes

$$y_k(\mathbf{x}, \mathbf{w}) = g^{(2)}\left(\sum_{j=0}^{M} w_{kj}^{(2)} g^{(1)}\left(\sum_{i=0}^{N} w_{ji}^{(1)} x_i\right)\right) \tag{2.36}$$

and is depicted in Fig. 2.9. The 3-layer feed-forward neural network is the most commonly used network structure in practice. This is partly due to the fact that a universal approximation theorem [106] says that any continuous function can be approximated to any given accuracy with a neural network having three or more nodal layers, guaranteed the activation functions are not

polynomials, are piecewise continuous and are locally bounded [88]. Thus a simple 3-layer network with sigmoidal activation functions, as discussed above, is complex enough to be successfully applied to most applications.

Choosing the number of hidden nodes in a neural network is an important decision to make. Too many nodes might lead to overfitting, while too few will result in the network making a sub-optimal approximation of the target function. Through the use of theoretical considerations [107] and by looking at empirical evidence [108] it has been found that one hidden layer (as said above) having $2N + 1$ nodes (N being the amount of inputs) is the optimal neural network structure for the approximation of continuous functions without the algorithm doing unnecessary work. This is further supported by the empirical studies carried out by Serra-Ricart et al. [109].

2.2.4.3 Autoencoders

We consider here very briefly a type of feed-forward network called an autoencoder, shown in Fig. 2.10. The main source of information for this section was the paper by Graff et al. [88] unless indicated otherwise.

In autoencoders the inputs, denoted in the figure as x_1, x_2 and x_3, are mapped back to themselves (x'_1, x'_2 and x'_3) such that the network is essentially trained to approximate the identity function. An autoencoder typically have more than one hidden nodal layer, and has a central layer about which the network is symmetrical - this central layer has fewer nodes than either the input or the output nodal layers. In the figure, the central layer nodes are denoted by z_1 and z_2, respectively. Autoencoders can be seen as consisting of two parts: the first part, called the *encoder*, maps the input variables to the central layer while the second part, the *decoder*, maps the central layer variables to the output layer variables approximating the inputs as well as possible.

The hidden layer variables z_1 and z_2 are nonlinear functions of the inputs and can be seen as a reduced set of features. Autoencoders, in fact, can be used in this way for carrying out nonlinear dimensionality reduction, forming generalisations of methods like principal component analysis (PCA, see section 2.3.1). It has been shown by Sanger [110] that a single-hidden-layer autoencoder with linear activation functions is equivalent to PCA.

2.2.4.4 Network Training

A simple method for determining the parameters $\mathbf{w}$ of the simple feed-forward neural network is by minimising the error function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} ||\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n||^2 \tag{2.37}$$

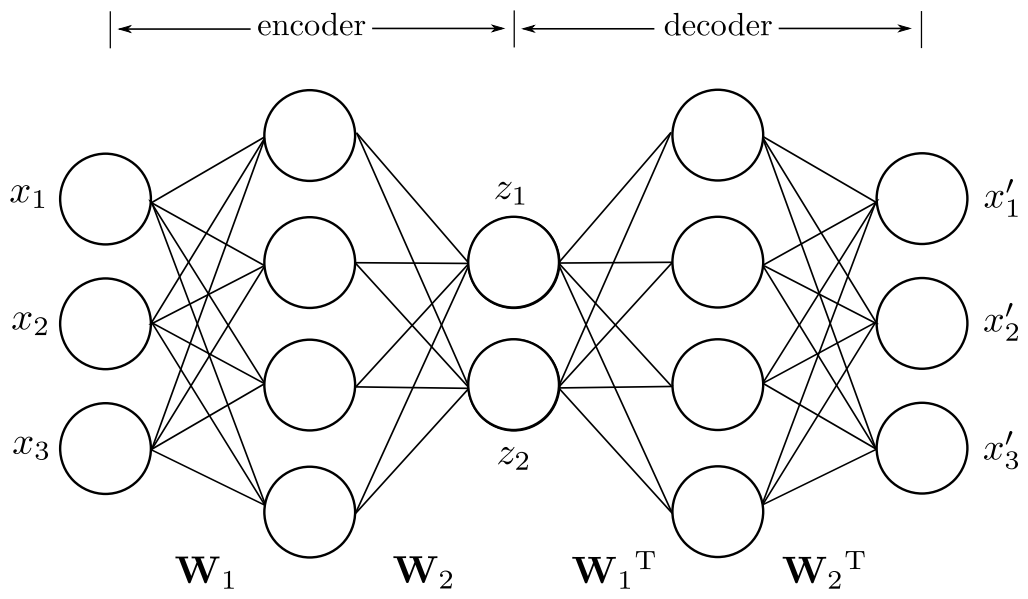


FIGURE 2.10: **An autoencoder.**
The inputs (x_1, x_2 and x_3) are mapped to themselves (x'_1, x'_2 and x'_3) via a network symmetrical about its central layer (depicted with nodes having the values z_1 and z_2). $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_1^{\mathrm{T}}$ and $\mathbf{W}_2^{\mathrm{T}}$ are the weights associated with the connections between the different nodal layers.

where $\{\mathbf{x}_n\}$, with $n = 1, \ldots, N$ is the set of input training feature vectors and where the set $\{\mathbf{t}_n\}$ consists of the corresponding target values. We show in Appendix B.1 how $\mathbf{w}$ is found in iterative steps by making use of the gradient of the error function. In a general case, the computation of the gradient is very expensive - however, in the case of a neural network, the gradient can be readily found via the backpropagation algorithm, which is described in Appendix B.2.

2.3 Feature Extraction

This section discusses principal component analysis (section 2.3.1) and linear discriminant analysis (section 2.3.2) in the context of feature extraction. It should be noted that these sections supply the standard theoretical background in order to provide a basis for understanding later chapters. Discussions and explanations of the methods are kept general, with the specific implementation of each of the algorithms and their deviations from the standard theoretical models discussed in Chapter 4.

2.3.1 Principal Component Analysis

Principal Component Analysis (PCA), also known as the *Karhunen-Loève* transform, is a technique widely used for tasks such as feature extraction, data compression and dimensionality reduction [92]. The main purpose of PCA is to reduce the dimensions of a data set having many correlated variables in such a way that most of the variance present in the original data set is preserved in

the new set. This is done by orthogonally transforming to a new set of uncorrelated variables, called principal components (PCs), ordered in such a way that the first few PCs contain most of the variance of the original data set [111].

Generally, it is accepted that Pearson (1901) [112] and Hotelling (1933) [113] gave the earliest definitions of PCA [111]. Both of these definitions are still commonly used today and both lead to the same algorithm. Hotelling's definition is much like the one given in the first paragraph, where a data set is projected orthogonally onto a linear space of lower dimension in such a way that the projected data's variance is maximised. This new linear space is known as the *principal subspace*. Pearson, on the other hand, defines PCA as the linear projection minimising the average projection cost, where this cost is defined as the mean squared distance between the original data points and their linear projections. [92]

The standard algebraic derivation of PCA, based on Hotelling's original definition, will be given in the following section.

2.3.1.1 Derivation of PCA – the Maximum Variance Formulation

Let $\{\mathbf{x}_n\}$ be a data set of observations, where $n = 1, ..., N$ and $\mathbf{x}_n$ is a D-dimensional Euclidean variable. The aim is to transform, or project, this data onto an M-dimensional linear space, where $M < D$, while at the same time maximising the projected data's variance.

We assume, for now, that M is known, and start off by projecting the data onto a one-dimensional space ($M = 1$), where the direction of the space is defined by a vector $\mathbf{u}_1$ of dimensionality D. Because we are only interested in the direction that $\mathbf{u}_1$ defines and not in its magnitude, we take it to be a unit vector, such that

$$\mathbf{u}_1^{\mathrm{T}}\mathbf{u}_1 = 1. \tag{2.38}$$

We then project each variable (data point) onto a scalar value as $\mathbf{u}_1^{\mathrm{T}}\mathbf{x}_n$, the mean of this projected data then being $\mathbf{u}_1^{\mathrm{T}}\overline{\mathbf{x}}$, where $\overline{\mathbf{x}}$ is the data set mean given by

$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n. \tag{2.39}$$

The variance of the projected data is then

$$\frac{1}{N}\sum_{n=1}^{N}\left\{\mathbf{u}_1^{\mathrm{T}}\mathbf{x}_n - \mathbf{u}_1^{\mathrm{T}}\overline{\mathbf{x}}\right\}^2 = \mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1 \tag{2.40}$$

where $\mathbf{S}$ is the covariance matrix of the data given by

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^{\mathrm{T}}. \tag{2.41}$$

To fully accomplish our goal, we next maximise the projected variance $\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1$, given in Eq. 2.40, with respect to $\mathbf{u}_1$. This is done in order to find the direction of projection $\mathbf{u}_1$ that would retain most of the variance present in the original data. In order to prevent $\|\mathbf{u}_1\| \longrightarrow \infty$, this maximisation has to be constrained, and we can do that by using the normalisation condition given in Eq. 2.38. In order to enforce this constraint, we use a Lagrange multiplier λ_1 so that we can then maximise the following:

$$\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1 + \lambda_1\left(1 - \mathbf{u}_1^{\mathrm{T}}\mathbf{u}_1\right) \tag{2.42}$$

Setting the derivative of Eq. 2.42 with respect to $\mathbf{u}_1$ equal to zero and taking the transpose of both sides, we find

$$\mathbf{S}\mathbf{u}_1 - \lambda_1\mathbf{u}_1 = 0 \tag{2.43}$$

or

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1. \tag{2.44}$$

From Eq. 2.44 it can be seen that $\mathbf{u}_1$ is an eigenvector of $\mathbf{S}$. Left-multiplying with $\mathbf{u}_1^{\mathrm{T}}$ and keeping in mind that $\mathbf{u}_1^{\mathrm{T}}\mathbf{u}_1 = 1$, we have the following:

$$\mathbf{u}_1^{\mathrm{T}}\mathbf{S}\mathbf{u}_1 = \lambda_1. \tag{2.45}$$

From Eq. 2.45 it can be seen that the projected variance is equal to the eigenvalue λ_1, and it is therefore evident that in order to maximise the variance we have to set $\mathbf{u}_1$ equal to the eigenvector with the largest eigenvalue (λ_1). This eigenvector is then called the first principal component.

Additional PCs can be defined by each time choosing a new direction (from amongst all the directions orthogonal to those already picked) that maximises the projected variance. In general, for an M-dimensional principal subspace, the optimal projection maximising the projected variance

will be defined by the M eigenvectors $\mathbf{u}_1, ..., \mathbf{u}_M$ of $\mathbf{S}$ corresponding to the M largest eigenvalues $\lambda_1, ..., \lambda_M$. This can be shown using proof by induction, as follows.

If we suppose that the above result holds for an M-dimensional principal subspace, an $M+1$ dimensional projection space will be defined by the M PCs $\mathbf{u}_1, ..., \mathbf{u}_M$ as well as an additional PC $\mathbf{u}_{M+1}$, whose value is to be determined here. In order to ensure that $\mathbf{u}_{M+1}$ is not linearly related to any of the already defined PCs $\mathbf{u}_1, ..., \mathbf{u}_M$, we set the constraint that it should be orthogonal to them (Eq. 2.46), and enforce it with Lagrange multipliers $\eta_1, ..., \eta_M$.

$$\sum_{i=1}^{M} \mathbf{u}_{M+1}^{\mathrm{T}} \mathbf{u}_i = 0 \tag{2.46}$$

Following the same arguments as for $\mathbf{u}_1$, we find that the projected variance is now given by $\mathbf{u}_{M+1}^{\mathrm{T}} \mathbf{S} \mathbf{u}_{M+1}$, and we maximise this using the Lagrange multiplier λ_{M+1} to enforce the constraint in the form of the normalisation condition given in Eq. 2.47.

$$\mathbf{u}_{M+1}^{\mathrm{T}} \mathbf{u}_{M+1} = 1. \tag{2.47}$$

Enforcing the constraints in both Eq. 2.46 and 2.47 using Lagrange multipliers, we now have to find the maximum of the function

$$\mathbf{u}_{M+1}^{\mathrm{T}} \mathbf{S} \mathbf{u}_{M+1} + \lambda_{M+1}(1 - \mathbf{u}_{M+1}^{\mathrm{T}} \mathbf{u}_{M+1}) + \sum_{i=1}^{M} \eta_i \mathbf{u}_{M+1}^{\mathrm{T}} \mathbf{u}_i. \tag{2.48}$$

By setting its derivative with respect to $\mathbf{u}_{M+1}$ equal to zero, we obtain the following:

$$2\mathbf{S}\mathbf{u}_{M+1} - 2\lambda_{M+1}\mathbf{u}_{M+1} + \sum_{i=1}^{M} \eta_i \mathbf{u}_i = 0 \tag{2.49}$$

By left-multiplying with $\mathbf{u}_j^{\mathrm{T}}$ and using the constraints for orthogonality, we find that $\eta_j = 0$ for $j = 1, ..., M$. Eq. 2.50 is therefore obtained.

$$\mathbf{S}\mathbf{u}_{M+1} = \lambda_{M+1}\mathbf{u}_{M+1} \tag{2.50}$$

From the above equation it can be seen that $\mathbf{u}_{M+1}$ is an eigenvector of $\mathbf{S}$ having eigenvalue λ_{M+1}. Left-multiplying with $\mathbf{u}_{M+1}^{\mathrm{T}}$ and keeping Eq. 2.47 in mind, we find that, similar to the case with $\mathbf{u}_1$, we have that the projected variance is given by

$$\mathbf{u}_{M+1}^{\mathrm{T}}\mathbf{S}\mathbf{u}_{M+1} = \lambda_{M+1}, \tag{2.51}$$

so that we can attain the maximum variance by selecting $\mathbf{u}_{M+1}$ as the eigenvector having the largest eigenvalue λ_{M+1} amongst those not already chosen. It is therefore seen that the result also holds for principal subspaces of dimensionality $M + 1$, completing the inductive step. Seeing as this has already been shown for the case where $M = 1$, if follows that the result must hold for any $M \leq D$.

In summary, PCA involves first finding the mean $\overline{\mathbf{x}}$ of the data set, then evaluating the data covariance matrix $\mathbf{S}$, and finally finding the M eigenvectors (PCs) of $\mathbf{S}$ corresponding to the M largest eigenvalues. To extract the features of an instance $\mathbf{x}_n$, it is reconstructed using a linear combination of the M largest eigenvectors, and the M corresponding coefficients can then be used as the instance's new features. Seeing as $M < D$, $\mathbf{x}_n$ has undergone dimensionality reduction. The M-dimensional vector of coefficients, also called PC weights, for a specific instance $\mathbf{x}_n$ is denoted by $\mathbf{a}_n$, and can be calculated as

$$\mathbf{a}_n = \mathbf{U}^{\mathrm{T}}(\mathbf{x}_n - \overline{\mathbf{x}}) \tag{2.52}$$

where $\mathbf{U}$ is a $D \times M$ dimensional matrix having the M largest eigenvectors (PCs) as columns. Here $\mathbf{a}_n$ is now the new feature vector of $\mathbf{x}_n$.

2.3.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is a technique that can be used for dimensionality reduction - it effectively projects data in the direction that will maximise the variance between respective classes while also minimising variance within classes, leading to greater class separation. The term *linear discriminant analysis* is often used interchangeably with the term *Fisher's linear discriminant (FLD)* due to the fact that these two methods, in spite of having slightly different derivations, are essentially the same - the difference lying only in that FLD does not make all of the assumptions LDA makes, such as equal class covariances and normally distributed class data. Because the derivation of FLD is more intuitive and explanatory, we will use it in this section - by making the assumptions LDA requires we arrive at the exact same result.

Considering the case of dimensionality reduction for two classes, our goal is to project the D-dimensional input instance $\mathbf{x}$ to one dimension as follows

$$y = \mathbf{w}^{\mathrm{T}}\mathbf{x} \tag{2.53}$$

We can ensure that this data projection maximises the separation of classes by adjusting the elements of $\mathbf{w}$, the weight vector, accordingly. If there are N_1 data instances in class $\mathcal{C}_1$ and N_2 in class $\mathcal{C}_2$, the two classes have means given by

$$\mathbf{m}_1 = \frac{1}{N_1}\sum_{n\in\mathcal{C}_1}\mathbf{x}_n \qquad \text{and} \qquad \mathbf{m}_2 = \frac{1}{N_2}\sum_{n\in\mathcal{C}_2}\mathbf{x}_n \tag{2.54}$$

The easiest way in which to acquire a measure of class separation (when the data is projected on $\mathbf{w}$), is to look at the separation of the class means of the projected data. We can therefore choose $\mathbf{w}$ such that it maximises

$$m_2 - m_1 = \mathbf{w}^{\mathrm{T}}(\mathbf{m}_2 - \mathbf{m}_1) \tag{2.55}$$

with

$$m_k = \mathbf{w}^{\mathrm{T}}\mathbf{m}_k \tag{2.56}$$

being the mean of the projected instances from class $\mathcal{C}_k$, where $k \in (1, 2)$ seeing as we are only considering a two-class system. As this expression can grow arbitrarily large if $\mathbf{w}$ is not constrained, we impose that $\mathbf{w}$ be of unit length, such that

$$\sum_i w_i^2 = 1 \tag{2.57}$$

Introducing a Lagrange multiplier and performing the maximisation, we then have that

$$\mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1) \tag{2.58}$$

There is a problem with this, though, as is shown on the left of Fig. 2.11. It can be seen here that whereas the two classes are originally well separated, there is considerable class overlap when the data is projected onto a line joining the class means. Fisher's idea [114] is in maximising a function giving a big separation between the means of the projected classes and a small variance

in every projected class, in this way minimising the overlap in classes. Eq. 2.53 projects labelled instances ($\mathbf{x}$) to labelled values in a one-dimensional space (y). The projected instances from class $\mathcal{C}_k$ therefore have a within-class variance of

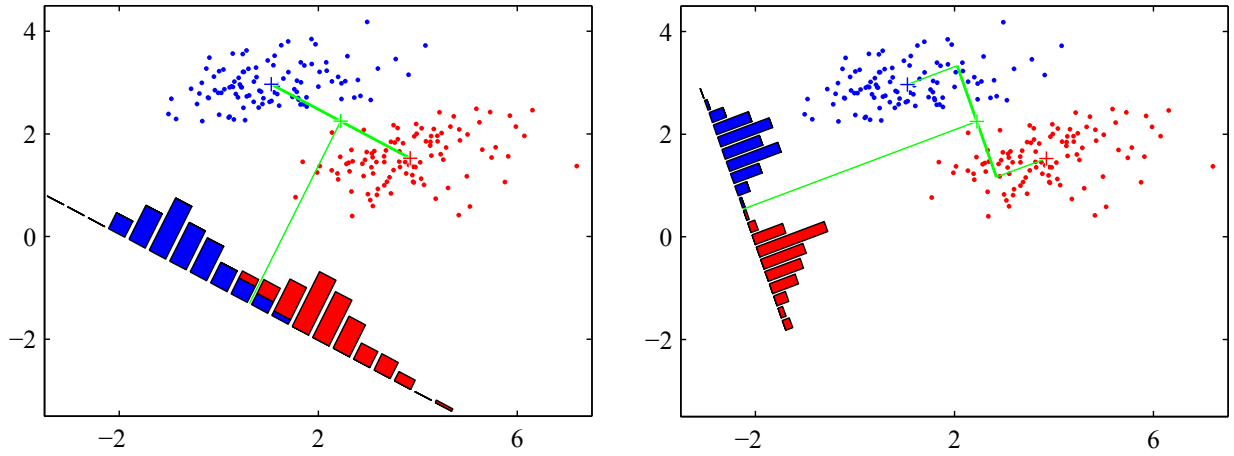


FIGURE 2.11: **LDA**
Left: An illustration of data instances from two classes (shown in different colours) and their projection on a line that joins the two respective class means, resulting in substantial overlap between the classes. **Right:** The Fisher linear discriminant projection, showing much better class separation. This figure was taken from [92].

$$s_k^2 = \sum_{n \in \mathcal{C}_k} (y_n - m_k)^2 \tag{2.59}$$

with y_n given by Eq. 2.53 as $\mathbf{w}^{\mathrm{T}}\mathbf{x}_n$. The within-class variance for the entire data set is then just $s_1^2 + s_2^2$. Next we define the Fisher criterion as the ratio of between-class to within-class variance, as

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \tag{2.60}$$

and then use Eq. 2.53, 2.56 and 2.59 to rewrite it as

$$J(\mathbf{w}) = \frac{\mathbf{w}^{\mathrm{T}}\mathbf{S}_{\mathrm{B}}\mathbf{w}}{\mathbf{w}^{\mathrm{T}}\mathbf{S}_{\mathrm{W}}\mathbf{w}} \tag{2.61}$$

with $\mathbf{S}_{\mathrm{B}}$, the between-class covariance matrix, equivalent to

$$\mathbf{S}_{\mathrm{B}} = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^{\mathrm{T}} \tag{2.62}$$

and $\mathbf{S}_\mathrm{W}$, the within-class covariance matrix, equivalent to

$$\mathbf{S}_\mathrm{W} = \sum_{n \in \mathcal{C}_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^\mathrm{T} + \sum_{n \in \mathcal{C}_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^\mathrm{T} \tag{2.63}$$

We next differentiate Eq. 2.61 with respect to $\mathbf{w}$ and set it equal to zero, after which it is easy to show that $\mathbf{w}$ maximises $J(\mathbf{w})$ when it satisfies

$$\mathbf{S}_\mathrm{B}\mathbf{w} = \lambda \mathbf{S}_\mathrm{W}\mathbf{w} \tag{2.64}$$

where λ is an eigenvalue. As we are only interested in the direction of $\mathbf{w}$, we note that $\mathbf{S}_\mathrm{B}\mathbf{w}$ has a direction equivalent to that of $(\mathbf{m}_2 - \mathbf{m}_1)$, leading to

$$\mathbf{S}_\mathrm{W}\mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1) \tag{2.65}$$

Multiplying with $\mathbf{S}_\mathrm{W}^{-1}$, we find

$$\mathbf{w} \propto \mathbf{S}_\mathrm{W}^{-1}(\mathbf{m}_2 - \mathbf{m}_1) \tag{2.66}$$

This result can now be used to project the data, using Eq. 2.53, in a way that leads to minimal class overlap. The above gives the exact same result as LDA if we assume that the class covariances are identical and the class data are normally distributed.

2.4 Concluding Remarks

The concepts, machine learning algorithms and feature extraction techniques discussed in this chapter supply the necessary theoretical background for later chapters. Background on the naïve Bayes classifier, the k-nearest neighbours algorithm, the support vector machine algorithm and artificial neural networks will be necessary for the discussions in Chapter 5, while Chapter 4 will require a knowledge of PCA and LDA.

Chapter 3

Supernova Cosmology

When I had satisfied myself that no star of that kind had ever shone before, I was led into such perplexity by the unbelievability of the thing that I began to doubt the faith of my own eyes.

– Tycho Brahe

3.1 Introduction

This chapter supplies the reader with the necessary supernova (SN) cosmology background and discusses the data that will be used in later chapters for machine learning purposes. Section 3.2 gives a brief overview of supernova cosmology and the problems associated with the vast amount of data that will be produced by the next generation of transient surveys, while section 3.3 discusses the Sloan Digital Sky Survey II (SDSS-II) SN Survey from which we draw the data used in this thesis. Finally, section 3.4 lays out the machine learning goals of this project, describes the dataset we use to accomplish it and briefly outlines the rest of the chapters in this thesis.

3.2 Supernova Cosmology

Supernovae, first explained in 1934 by Baade and Zwicky [115] following earlier work by Lundmark [116], are among the most violent events in the Universe. They occur at the end of a star's lifetime, when the star is destroyed by an explosion so powerful that it can briefly outshine an entire galaxy.

SNe are classified based on their spectral features at maximum light [117]. Type I SNe have no hydrogen in their spectra, while Type II SNe have. Type Ib SNe shows prominent helium lines; Type Ic displays neither hydrogen nor helium, and Type Ia SNe have spectra dominated by

lines from higher-mass elements like silicon, sulphur, iron and calcium while lacking hydrogen and helium [118].

Astronomically observed SNe correspond to two very different physical phenomena: the thermonuclear explosion of a naked white dwarf and the core collapse of a star. SNe of class Type Ia are of the thermonuclear type, and occurs in binary systems where two stars, a white dwarf and its companion star, orbit one another. A white dwarf is the latest stage in the evolution of some stars - at this stage, the envelope has been fully expelled and a degenerate core is all that remains of the star. As the companion star enters its red giant phase[1] it starts to inflate and expands beyond its Roche lobe[2], expelling matter that can then be accreted by the white dwarf. The mass of the white dwarf increases until it reaches the Chandrasekhar limit[3] ($\sim 1.4 M_\odot$, where $M_\odot$ denotes solar mass) after which it collapses. This collapse triggers explosive nuclear burning of the white dwarf which completely destroys the entire star, leaving no remnant in its wake.

FIGURE 3.1: **The Crab Nebula and Crab Pulsar.**
Left: The Crab Nebula - ejecta from the explosion. Credit: European Southern Observatory (ESO). **Right:** The Crab Pulsar in the centre of the Nebula that remained after the explosion. This is a superposition of an optical HST image (in red) and a false-colour Chandra x-ray image (in blue). Credit: Arizona State University (ASU), Hubble Space Telescope (HST), National Aeronautics and Space Administration (NASA). [119]

Even though Type Ia SNe are thought to be the thermonuclear explosion of a white dwarf, it is not clear that the mechanism under which the white dwarf aggregates mass from a companion star is the only way in which such a star can result in such a SN. The scenario under which two white

[1] All stars are born in the so-called main-sequence, but during their evolution they will leave the main sequence and enter the so-called red giant branch where they eventually become much larger and redder. This happens when a star runs out of hydrogen in its core, causing the core to contract and, as a consequence, its envelope to expand.

[2] In a binary system, the Roche lobe of a star is the region of space where material is gravitationally bound to that star.

[3] The Chandrasekhar limit is the mass beyond which the electron degeneracy pressure in the star cannot stop gravitational collapse. This pressure originates from the Pauli exclusion principle, which doesn't allow some particles to occupy the same quantum state.

dwarfs merge to produce a Type Ia SN is thought to be a sub-dominant mechanism to produce this type of SN and has been studied in Maoz, Mannucci & Nelemans [120] via simulations which have concluded that at least some Type Ia SNe could come from such a mechanism.

In the standard picture of core collapse SNe, on the other hand, they correspond to the spectral classes Type Ib, Ic and II and occur when the core of a massive star ($M \gtrsim 9\ \mathrm{M}_\odot$, where M denotes the mass of the star) can no longer withstand the gravitational pressure and collapses, releasing enormous amounts of gravitational energy. The imploded core can remain either as a neutron star[4] (usually in the form of a pulsar[5]) or a black hole[6], or it can be entirely destroyed, leaving no remnant. The outcome depends on the initial mass and metallicity of the star [119, 121]. An example of well-known SN remnants are the Crab Nebula and Crab Pulsar shown in Fig. 3.1, remains of the historical SN explosion that was recorder by the Chinese in 1054. Table 3.1 encapsulates the information given above.

TABLE 3.1: **Characteristics of different types of SNe.**

<table>
<tr><th>Spectral Type</th><th>Ia</th><th>Ib</th><th>Ic</th><th>II</th></tr>
<tr><td rowspan="3">Spectrum</td><td colspan="3">No Hydrogen</td><td rowspan="3">Hydrogen</td></tr>
<tr><td rowspan="2">Silicon</td><td colspan="2">No Silicon</td></tr>
<tr><td>Helium</td><td>No Helium</td></tr>
<tr><td>Physical Phenomena</td><td>Nuclear explosion of low-mass star</td><td colspan="3">Core collapse of evolved massive star (star may have lost its hydrogen or helium envelope during red-giant evolution)</td></tr>
<tr><td>Compact Remnant</td><td>None</td><td colspan="3">Neutron star (typically appearing as a pulsar), black hole or none.</td></tr>
<tr><td>Light Curve</td><td>Standardisable</td><td colspan="3">Large Variations</td></tr>
</table>

Although most known SNe can be readily classified as Type Ia, Ib, Ic or II, our knowledge of SNe is by far not complete, and we are discovering and postulating new types of SNe continuously. For example, a non-standard type of SN can arise when two white dwarfs collide, producing a thermonuclear SN [122]. As another example, there has been recent discoveries of superluminous supernovae (SLSN) about 50 times brighter than standard SNe types which are currently poorly studied due to a lack of data and are hence poorly understood. For a review, see [123].

[4] These stars are the smallest (with a radius of roughly $12 - 13$ km) and densest stars that we know of - they have a typical mass between $\sim 1.4\ \mathrm{M}_\odot$ and $3\ \mathrm{M}_\odot$.

[5] Pulsars are fast rotating, highly magnetised neutron stars. They emit electromagnetic radiation in a beam, which we detect in periodic pulses as the pulsar rotates. The intervals between pulses are very precise, ranging from roughly seconds to milliseconds for a given pulsar.

[6] A mathematically defined spacetime region having such a powerful gravitational pull that no radiation or particle can escape it.

Whereas there is a lot of variation in the light curves of Types Ib, Ic and II SNe, SNe of class Type Ia are surprisingly standardisable. For an example of how the light-curves of different SN types vary, see Fig. 3.2. The light-curves of Types Ib, Ic and II SNe differ from one-another due to the fact that they depend largely on the mass and metallicity of the progenitor star - and these tend to vary. Type Ia SNe, on the other hand, are typically brighter than Type Ib, Ic and II SNe and have light-curves that are very similar due to the fact that their progenitor stars are roughly identical - these SNe are used as "standard candles" for determining the luminosity distance to cosmological objects. Before discussing these standard candles, it is necessary to take a look at distance measurements in the Universe.

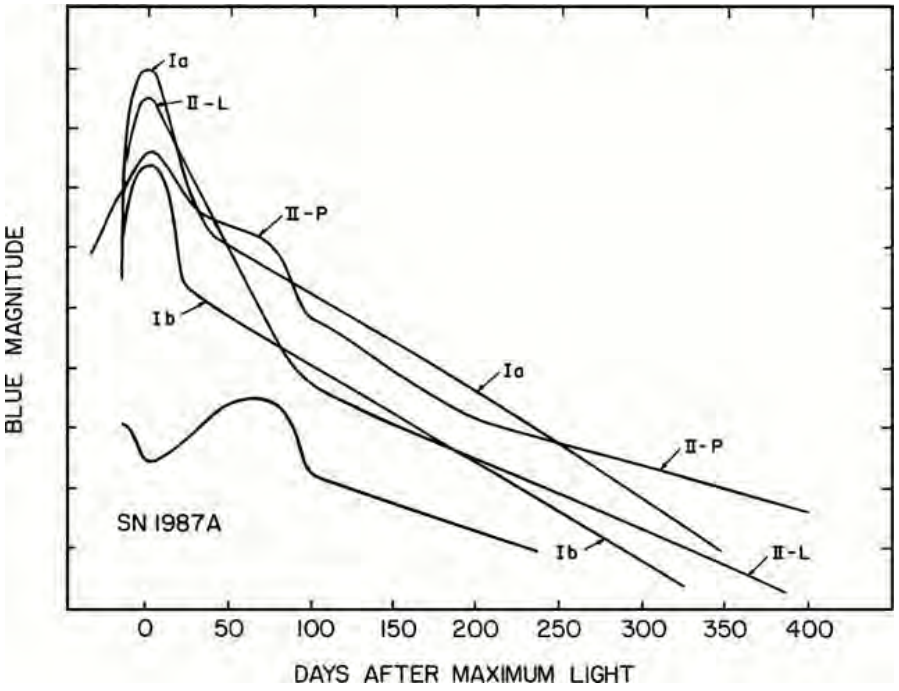


FIGURE 3.2: **Schematic light-curves of SNe.**
Schematic light-curves (linear in B-magnitude) for SN Type Ia, Ib, II-L, II-P and SN 1987A are shown here. The curve for Type Ib SNe includes Type Ic SNe as well, and represents an average of the two classes. SNe 1979C and 1980K are used for the Type II-L curve, although it should be noted that these may be unusually bright. [124, 125] For more information regarding SNe light-curves, see [126–129].

3.2.1 Distance Measurements

This section looks at the derivation of the luminosity distance. In order to do this, we start by outlining how the metric[7] changes in an expanding universe. The Robertson-Walker (RW) metric describes such an expanding, isotropic and homogeneous universe and is given by

[7]The metric gives the distance in a space between points differing by an infinitesimal coordinate difference dx_μ and controls whether the space is curved or not. μ is a spacetime direction.

$$ds^2 = c^2dt^2 - a(t)^2\left[\frac{dx^2}{1-\kappa x^2/R_0^2} + x^2 d\Omega^2\right], \tag{3.1}$$

where the terms inside the square brackets represent the spatial part of the metric which is being stretched as $a(t)$, the scale factor[8], changes. This metric allows non-Euclidean[9] geometry, where $\kappa = +1$ if the Universe is closed (spatial sections are 3-spheres), $\kappa = -1$ if the Universe is open (spatial sections have hyperbolic geometry), and $\kappa = 0$ if the Universe is Euclidean and flat, with R_0 denoting the radius of curvature. $d\Omega$ is a solid angle, and corresponds to the angular part of the metric, t denotes time, c is the speed of light, ds is the spacetime distance and x is the radial comoving distance.

It is convenient and common to make the following coordinate transformation

$$x = S_\kappa(r) = \begin{cases} R_0 \sin(r/R_0) & \text{if} \quad \kappa = +1 \\ r & \text{if} \quad \kappa = 0 \\ R_0 \sinh(r/R_0) & \text{if} \quad \kappa = -1, \end{cases} \tag{3.2}$$

so that the metric can be written as

$$ds^2 = c^2dt^2 - a(t)^2\left[dr^2 + S_\kappa(r)^2 d\Omega^2\right]. \tag{3.3}$$

If we want to calculate the comoving distance between us and an object that has emitted its light at time t when the Universe had a scale factor of a, we need to integrate the radial part of the metric. The theory of relativity tells us that ds is equal to zero for the path of a photon. Therefore, if we neglect the angular part of the metric and set $ds = 0$, we can write

$$d_c(a) = S_\kappa\left(\int dr\right) = S_\kappa\left(\int \frac{cdt}{a(t)}\right) \tag{3.4}$$

where $d_c(a)$ is the comoving distance. Usually, this integral is written as a function of the Hubble parameter, which is defined as $H(a) = \dot{a}/a$. The Hubble parameter is what relates the recession velocity of galaxies to the distance of such galaxies from us, and changes as a function of time. The Hubble parameter today is called the Hubble constant H_0. The parameter can be derived

[8] The scale factor is a multiplicative factor that describes how much the Universe has changed in size from time t to today.

[9] Given that Einstein proved that mass bends space, the spatial part of the metric assumes that the Universe can also be curved, hence non-Euclidean.

by writing the solution of Einstein's equations with the RW metric. This results in the Friedman equation, given below, and depends on the contents of the Universe.

$$H(a)^2 = H_0^2 \left(\frac{\Omega_m}{a^3} + \frac{\Omega_\kappa}{a^2} + \Omega_\Lambda \right) \tag{3.5}$$

In Eq. 3.5 Ω_m denotes the ratio of the density of mass in the Universe to the density of mass required for a flat Universe, $\Omega_\kappa = -\kappa c^2/(R_0 H_0)^2$ and Ω_Λ is the ration of the density of dark energy in the Universe to the density of mass required for a flat Universe. We therefore find

$$d_c(a) = S_\kappa \left(\frac{c}{H_0} \int \frac{a^{-2} da}{\sqrt{\Omega_m a^{-3} + \Omega_\kappa a^{-2} + \Omega_\Lambda}} \right). \tag{3.6}$$

In a non-expanding universe, the comoving distance $d_c(a)$ and the luminosity distance $d_L(a)$, which is the distance one can infer by measuring the flux of a source of known luminosity, are the same. In an expanding universe, however, they are not. The energy of photons is decreased by a factor of a as they are stretched while travelling from their source to us. The rate at which such photons arrive is also decreased by another factor of a as we are moving away from the source due to the expansion of the universe. Therefore,

$$d_L(a) = \frac{d_c(a)}{a}. \tag{3.7}$$

We have therefore derived the luminosity distance and shown that it depends on the cosmological parameters Ω_m, Ω_κ and Ω_Λ.

3.2.2 Type Ia Supernovae as Standard Candles

SNe Ia can be used as standard candles for the determination of the luminosity distance to astronomical objects. Standard candles are objects with very well known absolute magnitudes[10] (related to the intrinsic brightness of an object). By comparing such an object's apparent magnitude with that of its absolute magnitude, its luminosity distance can be calculated.

Let us consider a cosmological source with flux F and known luminosity L - the received flux then decreases with distance d_L from the source as follows

[10]The magnitude system is used to indicate the brightness of objects - the larger the magnitude, the fainter the object. A difference of one magnitude is equivalent to a brightness difference of $\sqrt[5]{100}$. The *apparent magnitude* of an object is its magnitude as observed from Earth, while its *absolute magnitude* is the hypothetical apparent magnitude an object would have at a standard luminosity distance of 10 pc.

$$F = \frac{L}{4\pi d_L^2} \tag{3.8}$$

Working with fluxes leads to the concept of an object's magnitude, as the luminosity distance d_L of an object can be described in terms of its apparent magnitude m and its absolute magnitude M, as

$$\mu = m - M = 5 \log d_L + 25 \tag{3.9}$$

The luminosity distance d_L is here measured in Mpc, and μ is the object's distance modulus. The luminosity distance of an object can therefore be determined if its apparent and absolute magnitudes are known.

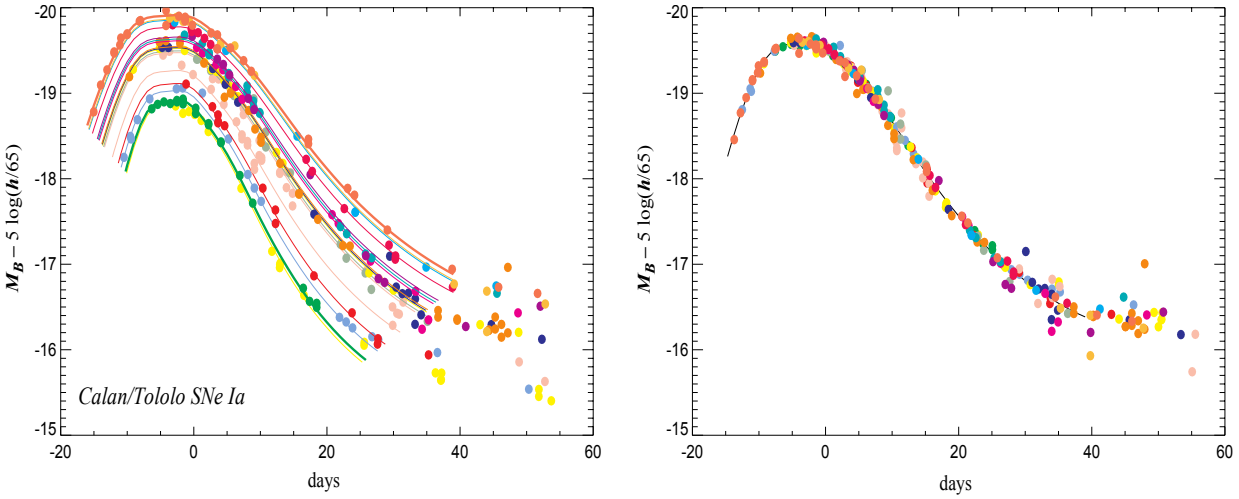


FIGURE 3.3: **Standardisation of SNe lightcurves.**
Light curves of a sample of SNe Ia discovered by the Calan/Tololo Supernova Survey before (left) and after (right) standardisation [130].

SNe Ia can be used as standard candles due to certain assumptions concerning the mass involved in each explosion and the main characteristics of the explosions. As mentioned before, the progenitor stars are all white dwarves in binary systems expected to explode at the Chandrasekhar mass. This similarity in mass results in there being a uniformity in energy released during the supernova explosion, and therefore also a uniformity in intrinsic luminosity (related to the absolute magnitude) of the sources. Even more importantly, a correlation between their decay times and their peak brightness exists, known as the Phillips relation [131–133], allowing us to standardise the sources and greatly reduce the variation within the class of Type Ia SNe. This enables us to very precisely determine their absolute magnitudes which, in turn, reduces the distance moduli (and luminosity

distance) estimation errors for the SNe (see Eq. 3.9). Other corrections to the SN magnitude include correcting for dust extinction as well as transferring the observed flux to the rest frame using a K correction term[11] [117]. See Fig. 3.3 for an example of this standardisation. [119]

As was seen in Eq. 3.7, the luminosity distance is related to cosmological parameters. Many cosmological models, for example, are fitted to the distance modulus μ, with μ a function of redshift z, in order to constrain their cosmological parameters. Redshift is related to the change in frequency ν that a photon emitted at a scale factor a_1 undergoes up to a scale factor a_2, and can be expressed as follows:

$$z = \frac{\nu_1 - \nu_2}{\nu_2} = \frac{a_2}{a_1} - 1. \tag{3.10}$$

Since $a = 1$ today, we can write down a useful relationship between the redshift of an object observed in the Universe and the scale factor as

$$a = \frac{1}{z+1}. \tag{3.11}$$

This section therefore concludes that SNe Ia are excellent cosmological probes. The use of SNe Ia as standard candles by Perlmutter et al. [134] and Riess et al. [135] resulted in the first conclusive evidence of the accelerated expansion of the Universe. This was indicative of the existence of dark energy, and earned them (together with Brian Schmidt) the 2011 Physics Nobel Prize.

3.2.3 The Data Deluge

Due to its importance in the constraint of cosmological parameters, supernova cosmology is a well-established field seeing continuous development, with surveys such as the Large Synoptic Survey Telescope (LSST)[12], the Dark Energy Survey (DES)[13], the Panoramic Survey Telescope & Rapid Response System (Pan-STARRS)[14] and the Palomar Transient Factory (PTF)[15] set to produce a vast amount of data in the years to come.

However, such a data deluge will require new statistical inference and machine learning techniques for processing and analysis. The LSST, as an example, will image the entire southern sky every

[11] A K correction is a redshift- and filter-dependent correction to the magnitude of a given object that arises from the fact that an object's spectrum is shifted towards the red when the object is at a higher redshift; hence, for a set of objects with the same rest-frame spectrum placed at different redshifts, a given filter will sample different parts of the rest-frame spectrum of each object without this correction.

[12] http://www.lsst.org

[13] http://www.darkenergysurvey.org/

[14] http://pan-starrs.ifa.hawaii.edu/public/

[15] http://www.astro.caltech.edu/ptf/index.php

night with such frequency and depth that the amount of transient alerts expected each night is more than a million, while on the time-scale of a decade at least a million Type Ia SNe candidates will be detected [136]. Existing spectroscopic follow-up capabilities will be swamped by this amount of data, and we will enter an era where small spectroscopic subsets will be used to train processes of photometric transient identification [137–143]. Such methods will always result in a small number of misidentifications, with the danger being that this contamination, unless dealt with carefully, can lead to biased results.

This vast amount of data creates problems in the analysis pipeline long before the final scientific analysis is reached, however. As an example, consider difference images, produced by the subtraction of a reference image of a certain part of the sky from the latest image of that same area. Ideally, the reference image will consist of pure noise unless a transient like an asteroid, SN or variable star exists in the reference image. In real life, however, instrumental effects (e.g. registration errors, diffraction spikes, bleeding and CCD[16] saturation) cause the occurrence of artefacts.

It is therefore necessary to be able to distinguish between artefacts and objects that might potentially be of interest. This classification into artefacts and real objects has, historically, been carried out by astronomers scanning the images by hand shortly after they have been taken. For the Sloan Digital Sky Survey (SDSS) SN Survey[17], this practice lead to hand scanners typically sorting hundreds or thousands of images per night - a very time-consuming job. It has recently been shown that scanning such images by hand can be done quite successfully by crowdsourcing [144] - 93% of the SNe that was spectroscopically confirmed was correctly identified by the public.

Although this success sounds alluring, one should be reminded that using human beings for the purposes of classification introduces great difficulties in quantifying the biases originating from the different internal decision trees and algorithms in the brain of each scanner. Additionally, a scanner's effective decision tree will vary with time according to tiredness, mood and environment[18], making it impossible to be characterised systematically. For the SDSS-II SN Survey, the bias introduced by the scanners was partly dealt with by the injection of fake SNe into the system so that an average detection efficiency could be determined for every hand scanner - however, this is very clearly a limitation fundamental to human hand scanning that will be even worse in the case of crowdsourcing. Apart from this, hand scanning will be completely infeasible for surveys like the LSST due to the fact that millions of images will have to be scanned per night.

To succeed in the objectives of future transient surveys, the successful substitution of human hand scanners with machine learning techniques for the purpose of this artefact-transient classification

[16] A charge-coupled device (CCD) is a circuit etched onto a surface of silicon that forms light sensitive elements referred to as pixels. Electric charge is generated when photons hit the surface, and this is interpreted and/or manipulated by electronics and digitised.

[17] The SDSS: http://www.sdss.org/ and the SDSS-II SN Survey: http://classic.sdss.org/supernova/aboutsupernova.html.

[18] This is a well-known problem that affects even the most experienced practitioners. For example, the verdicts of judges vary strongly depending on the time since the last break [145].

therefore represents a vital frontier. Existing work comprise of work done by the SNfactory [146, 147], where the main object's shape, FWHM (Full Width at Half Measure) and position, along with the distance to the closest object also contained in the reference image were used as features. The work of the Palomar Transient Factory is also of interest, where the emphasis is on the separation of variable stars and transients [148] and the discovery, by [149], of variability in imaging surveys in the time domain where estimations of the degree to which newly found objects are astronomically relevant objects of variable brightness is output by classifiers in the form of probabilistic statements.

3.3 The Sloan Digital Sky Survey: Supernova Survey

The SDSS began its survey operations in 2000 and has progressed through several phases: SDSS-I (2000-2005), SDSS-II (2005-2008), SDSS-III (2008-2014) and SDSS-IV (2014-). Each of these phases comprised of multiple surveys having interlocking scientific goals.

The original SDSS observing plan that stretched from 2000 to 2008 (consisting of both SDSS-I and II), is known as the SDSS Legacy Survey - it resulted in a well-calibrated, uniform map of the Universe that will be used for scientific studies for decades to come. During the first five years of operation, SDSS-I carried out deep multi-colour imaging over 8000 deg^2 and measured spectra of over $700,000$ objects. SDSS-II completed the original survey goals of imaging half the northern sky as well as mapping the three-dimensional clustering of $100,000$ quasars and a million galaxies. Additional to this, SDSS-II carried out two more surveys: the Supernova Survey and the Sloan Extension for Galactic Understanding and Exploration (SEGUE). The Supernova Survey discovered hundreds of supernovae for the purpose of measuring the expansion history of the Universe, and it is this survey that is of interest to us.

3.3.1 The SDSS-II Supernova Survey

The SDSS-II SN Survey was a three-year project running from 1 September to 30 November each year from 2005 to 2007 with the aim of discovering and producing well-measured light curves of SNe Ia in an intermediate redshift range ($0.05 < z < 0.35$). The survey, using the SDSS telescope located at the Apache Point Observatory in New Mexico, did repeated five-band ($ugriz$) imaging of SDSS Stripe 82, a 300 deg^2 area centered on the celestial equator. [150]

The SN Survey was designed to address two main issues: the scarcity of SN Ia data found at intermediate redshifts (often referred to as the "redshift desert"), and the systematic limitations introduced by previous SN Ia surveys. At the time, cosmological constraints were based on a Hubble diagram constructed from low- ($z \lesssim 0.1$) and high-redshift ($z \gtrsim 0.3$) SN Ia samples, observed with a variety of telescopes, photometric passbands and selection criteria - naturally

introducing systematic errors. Obtaining more precise cosmological constraints therefore required more control over the sources of these systematic errors - calling for larger SN samples having continuous redshift coverage of the Hubble diagram that consisted of high-quality, multi-band data with well-understood photometric calibration. [150]

With its large areal coverage, photometric accuracy and sensitivity, the SDSS-II Supernova Survey could find thousands of SNe and transient objects at the intermediate redshift ranges lacking in SN Ia data. What is more, the survey was able to carry out a median of 10 single-epoch exposures per band for SNe, resulting in more detailed light-curve data that further reduced systematic errors. By surveying Stripe 82, it could take advantage of the extensive database of object catalogues, reference images and photometric calibration previously obtained by the SDSS-I for this same region. The uniformity of the instrumentation employed by both these surveys, in conjunction with the fact that both surveys obtained photometric standards from the same telescope, further minimised systematic errors. [150]

3.3.1.1 The Supernova Survey Science Goals

What follows is a short summary of the primary science goals that the SDSS-II SN Survey had, as set out by Frieman et al. [150].

1. **Refining the SN Ia Hubble diagram.** SN Ia distance measurements are required to constrain the history of the Hubble parameter. As the SDSS-II SN Survey was set to obtain distance modulus estimates for SN Ia in the redshift range $0.05 < z < 0.35$, a region that was up to that point sparsely populated on the Hubble diagram, it would provide data leading to the obtaining of information on the evolution of the cosmic scale factor that no other supernova survey could yet supply.

2. **Minimisation of SN systematics.** Because of the fact that, at the time, most SN surveys had systematic errors comparable to their statistical errors, the SDSS-II SN Survey aimed at improving control over systematic errors. The SDSS's photometric calibration errors are small due to the fact that effort were made over many years on the large-scale calibration of the data [151] - photometric errors of 1% have been achieved over Stripe 82, the area that the SDSS-II SN Survey observed [152]. Furthermore, the *ugriz* filters used by the SDSS-II SN Survey provided important colour information, and it used a stable, single camera with well-calibrated and well-measured filter transmission curves. The SDSS native magnitudes are also close to the AB system [153], with offsets that are well-measured.

3. **Anchoring the Hubble diagram.** It was estimated that the size and quality of the low-redshift portion ($z \lesssim 0.15$) of the SDSS-II SN Survey, upon completion, would be sufficient

to anchor the Hubble diagram and retrain light-curve fitters. This would reduce the necessity to rely on the heterogeneous low-redshift samples used up to that stage.

4. **Rest-frame ultraviolet light curve templates.** SNe surveys operating in the region $z > 1$ must have their SN light curves matched to low-redshift templates in order to reduce systematic errors. As an example, observations made at $z = 1.2$ in the reddest optical passbands ($\sim$ 8000 Å) correspond to 3600 Å(the u-band) in the SN rest-frame. At $z = 0.3$, the SDSS would observe this region at 4700 Å, the survey's g-band. The SDSS-II SN Survey would therefore be able to improve the rest-frame ultraviolet template data [154] necessary for interpreting high-redshift SNe data.

5. **SN characteristics from multi-band photometry.** Because spectra, which are used to determine SN types and redshifts, are obtainable for only a small amount of possible candidates, the SDSS aimed at improving measures to obtain this same information via other routes. The multi-coloured light curves traced out by SNe Ia as they evolve, in combination with their host-galaxy colours, can provide information on SN type, redshift and age.

6. **SN types, rates and host galaxies.** Apart from spectroscopically confirmed SNe Ia, the survey would also yield light-curves of additional SNe - objects likely to be SNe Ia, but is spectroscopically unconfirmed, as well as other types of SNe. This, in combination with the fact that the detection efficiency was monitored by inserting artificial SNe into the data stream, enabled the survey to obtain more robust measures of SN rates. Furthermore, the SDSS supernova sample would provide a large set of host galaxies whose characteristics can be used to glean information about the progenitor properties ([155, 156]). Lastly, because of the large volume that it covered, the SDSS-II SN survey would serve as a probe for rare and interesting objects.

3.3.1.2 Technical aspects of the SN Survey

The SDSS produced photometric measurements in each one of the *ugriz* SDSS pass-bands [157] using the 2.5m SDSS telescope [158] and photometric camera [159], covering the wavelength range from 350 to 1000 nm. The g, r and i bands are the best-suited for observing SNe, as the u and z bands have relatively poor throughput at low redshifts ($z \lesssim 0.1$ for SNe Ia). [160]

The SN Survey repeatedly scanned Stripe 82, an area 2.5° wide in Declination that stretches between a Right Ascension of 20^h and 04^h, in order to discover new SNe and take measurements of those already discovered. The camera was operated in drift-scan mode. Data in a CCD are read by sequentially shifting it from its current pixel to the pixel directly below until it reaches the end of the CCD where it is read out by external electronics and converted into a usable format. If the speed at which data is shifted in this manner is made equivalent to the speed at which an object in

the sky drifts by in front of the camera due to the Earth's rotation, the telescope is being operated in drift-scan mode. This enables the telescope to image long continuous strips of the sky.

The SDSS camera contains six columns of CCD chips - each column contains five chips, each with a different filter. The telescope (being operated in drift-scan mode) lets objects in the sky drift across these columns, and the CCD's are then read out in sync with the drifting. An illustration of the camera can be seen in Figure 3.4 - using the orientation in the figure, a specific region of the sky will drift vertically downward through a single column of the camera (as shown by the long arrows), encountering, in turn, the r, i, u, z and g filters. An object takes 55 seconds to drift across one CCD chip - the exposure time for each passband is therefore 55 seconds. By offsetting the camera slightly to account for the gaps between the columns of CCD chips, full coverage of Stripe 82 could be obtained in two nights, with the average cadence being roughly four nights due to weather effects and moonlight interference.

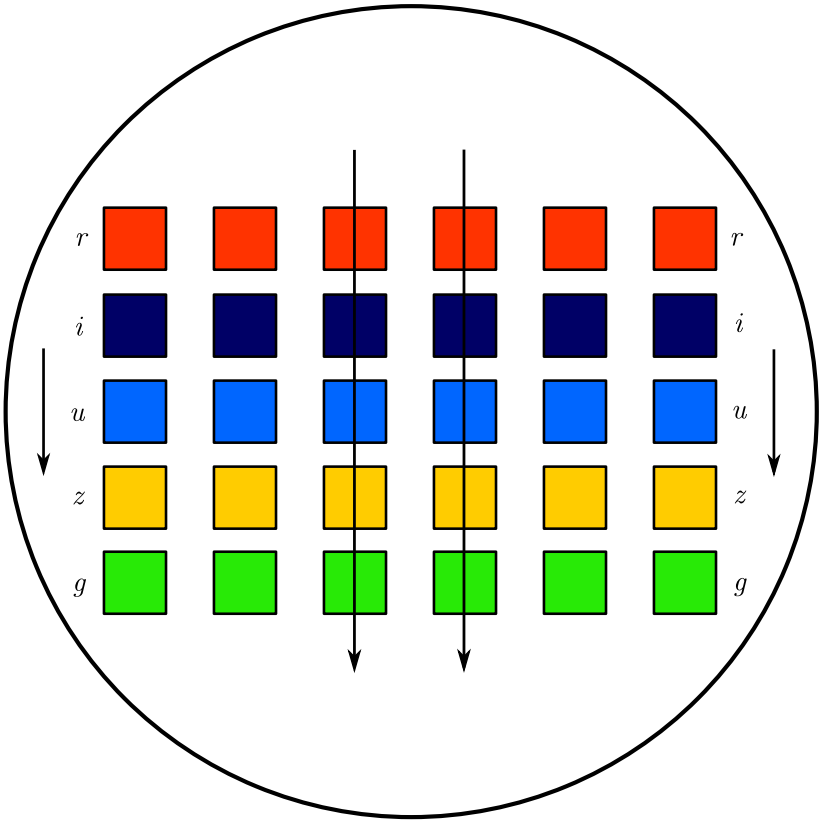


FIGURE 3.4: **The SDSS photometric camera.**
The camera consists of six columns of CCD chips - each column contains five chips, each with a different filter. Using the configuration in the figure, an object will drift vertically downward through a column (as shown by the arrows), in this way encountering each of the five SDSS filters (r,i,u,z and g) in turn.

The u, g, r, i and z-band SDSS camera images were processed using the first stages of the SDSS reduction pipeline [161] in order to produce calibrated images to be used as SN search data. Deep, co-added reference images (called templates), created using data taken throughout 2004 during

the SDSS-I Survey, were then subtracted from their corresponding search frames [162] to produce difference images for the purpose of SN detection. "Real-time" image subtraction was only carried out in the g, r and i bands, as they are most useful for SN detection.

The difference images were next sent through an automated object detection algorithm - objects that were detected in more than one of the three passbands, that didn't coincide with existing catalogued stars or variable objects and were not determined to be moving during the interval between the r and g exposures, were kept for further investigation. In 2005, these selected images were then sent to hand scanners who manually evaluated and classified the objects (see section 3.4.1).

In 2006 and 2007, however, additional software cuts were introduced in order to reduce the amount of images that required manual hand scanning. Now, single-epoch detections were only sent onwards to manual scanners if they were not detected as moving (with an improved algorithm, see [163]) and if they were bright enough ($r < 21$ or $g < 21$)[19]. Otherwise, detection in at least two epochs were required for an object to be sent for hand scanning. During visual hand scanning, an object was then denoted a "candidate" if it was deemed a possible SN, and some of these promising candidates were then sent for spectroscopic follow-up in order to determine whether they truly were SNe.

3.4 Classification Goals and Chosen Dataset

Following on from the big data problems of future surveys and human hand scanner biases discussed in section 3.2.3, the purpose of this thesis is to explore the possibility of successfully replacing human hand scanners in optical SN surveys with machine learning algorithms for the task of distinguishing between artefacts and transient[20] objects.

Existing work in this field has been discussed in section 3.2.3. In this thesis, however, we use data from the SDSS-II SN Survey and derive our features from Principal Component Analysis and Linear Discriminant Analysis (PCA and LDA, see Chapter 4) of the Sloan g, r and i difference images before comparing a number of different potential machine learning algorithms (see Chapter 5) with one another. The following subsections describe the dataset we use, as well as the classification systems we employed for use in our feature extraction methods and machine learning applications in later chapters.

[19] For the SDSS survey rate yielding a 55 second integrated exposure in each of its passbands, a 50% detection completeness is achieved for sources at $u = 22.5$, $g = 23.2$, $r = 22.6$, $i = 21.9$ and $z = 20.8$ [150, 164]. A source at $r = 21$ and $g = 21$ is therefore 1.6 magnitudes brighter in the r-band and 2.2 magnitudes brighter in the g-band than the detection threshold stated above.

[20] The word "transient" is here meant in an over-arching manner - it applies to all the real objects in our dataset, discussed in section 3.4.1 and further described in Table 3.2. It should not be confused with the similarly-named hand-scanner class discussed in the next section.

3.4.1 Dataset and Classification Systems

The data used in this thesis is taken from the last two years (2006 and 2007) of the SDSS-II SN Survey. Data from 2005 were omitted due to the different threshold cuts employed by the survey at the time (as mentioned in section 3.3.1.2) - we did not want to introduce unnecessary variation from outdated filtering procedures into our dataset. A team of roughly twenty hand scanners using the g, r and i-band search and difference images (each having a size of 51×51 pixels) as well as object history data classified each of the candidate objects into one of ten possible classes: dipoles, artefacts, saturated stars, transients, variables, moving objects, SN Gold, SN Silver, SN Bronze and SN Other [150]. Each one of these classes is described [165] in Table 3.2, and Fig. 3.5 depicts a simplified decision tree for the visual classification process followed by the hand scanners. Table C.1 shows different r-band object images for each of the above-mentioned classes, and also shows how image quality differ depending on the Signal to Noise Ratio (SNR). Table C.2 shows the g, r and i-band images of objects from each of the above-mentioned ten classes in order to show how images differ across bands. Our dataset consists of the g, r and i-band 51×51 pixel difference images classified by the hand scanners.

For our feature extraction purposes, we re-categorize the original ten classes into three visual classes, following the observation that some of the classes have visual appearances that are very similar. These three visual classes are: "artefacts", "real" objects and "dipoles/saturated", as illustrated in Fig. 3.6. The residuals of real objects are point-like (convolved with the telescope and atmosphere's seeing), artefacts have residuals resembling diffraction spikes and the residuals of dipoles/saturated objects are often quite point-like, typically with negative flux in some part of the image stemming from saturated CCD effects or registration errors. To see which original classes correspond to which visual classes, see Table 3.2. The visual classification system is used for the extraction of a certain set of features using PCA, as described in Chapter 4.

We are, however, ultimately interested in whether or not we can match the performance of human hand scanners in distinguishing between "real" objects (transients) and "not-real" objects (artefacts and dipole/saturated objects), and these are therefore the classes recognised by our machine learning classifiers - they are trained with object instances classified as either real or not-real. To see the classes corresponding to real and not-real objects, respectively, see Table 3.2.

3.4.2 Training and Testing Protocol

Our full dataset contains 27,480 objects of which 11,959 are not-real and 15,521 are real, where each object is associated with three 51×51 pixel images (one from each of the g, r and i-band). Of these, 2500 objects are fake SNe inserted into the pipeline for the characterisation of the selection

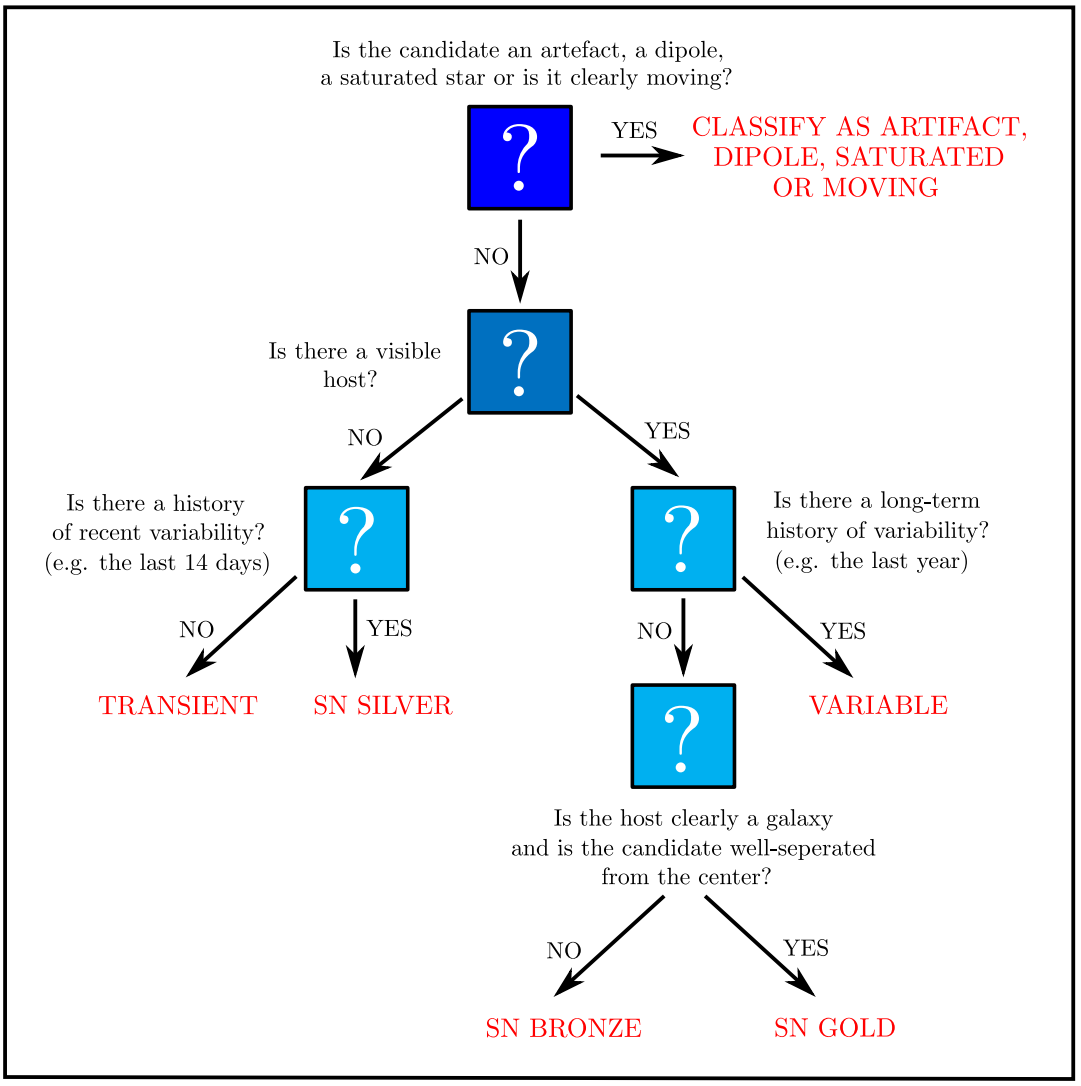
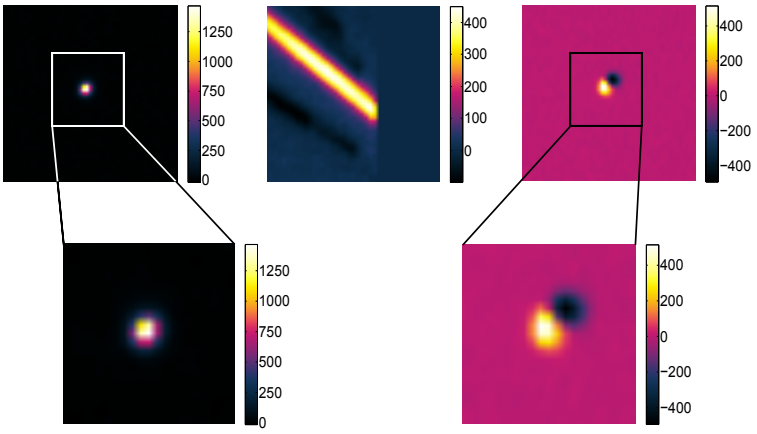


FIGURE 3.5: **Simplified hand-scanning flow-chart.**
A simplified decision tree depicting the visual classification process followed by the SDSS-II SN Survey hand scanners [165].

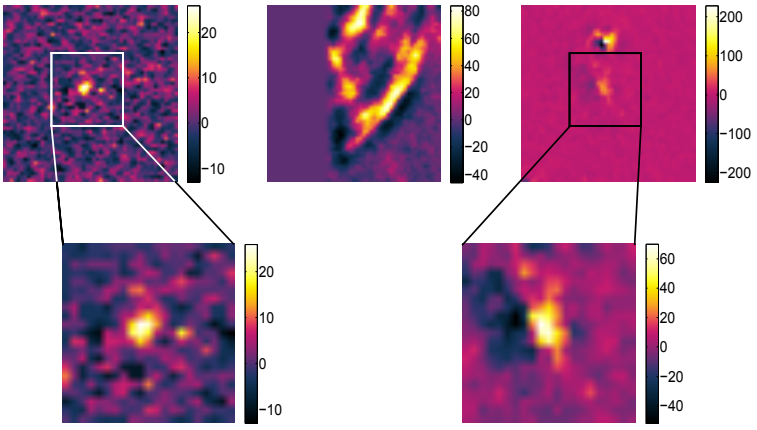
function and to provide quality control. Our dataset includes the fake SNe to allow for comparison of our results with the performance of humans on the fakes.

Our testing protocol was to sequester 25% of the full data set (referred to as the "test set") for algorithm comparison after the various classifiers have been trained and have undergone preliminary testing with the remaining 75% of the data. This data split was done before the various classifiers were built and the test set was kept apart from the rest of the data until we were ready for final testing.

Some of our learners can be optimised through the use of cross-validation - 30% of the remaining 75% mentioned above was used as a "validation set" to perform this optimisation while the remaining data, referred to as the "training set", were used for the training of the classifiers. The grouping of the data for these various purposes can be seen in Fig. 3.7.



(A)



(B)

FIGURE 3.6: **Visual classification.**
Example of the three different visual classes in the r-band; (A) shows high-quality images from the real (left), artefact (middle) and dipole/saturated (right) classes - all images here have an SNR above 40; (B) shows more representative images from the real (left), artefact (middle) and dipole/saturated (right) classes - all images here have an SNR below 20, representing roughly 80% of the images in the dataset.

It should be noted here that the initial isolation of the test set is absolutely necessary for an unbiased comparison of our various learners. If the same data is used for the optimisation of an algorithm than for the testing of it there is the danger of "training to the test set", something that almost certainly will lead to poor results in the real world due to overfitting.

3.4.3 Layout of the Remainder of the Thesis

Now that the necessary background knowledge has been supplied (Chapter 1, 2 and 3) and the objectives of the thesis has been discussed (section 3.4), the remainder of the chapters will be devoted to achieving these objectives. Chapter 4 will discuss the various techniques we employ for feature extraction, while Chapter 5 will detail the training, validation and testing of our five

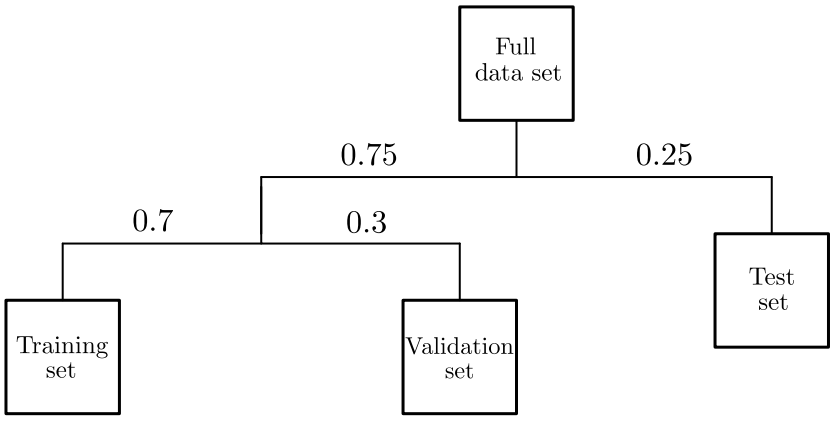


FIGURE 3.7: **Training, test and validation set.**
This diagram shows how the full data set was split for various purposes. The test set was sequestered until the final testing stage, and the remaining data was split into a further training and validation set. The validation set was used for parameter optimisation of certain classifiers.

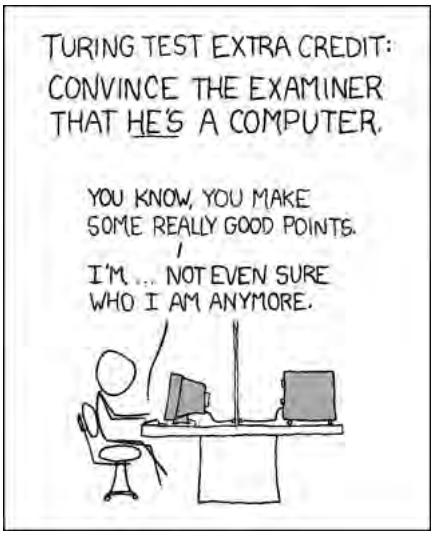
chosen machine learning algorithms and discuss their results. Finally, our conclusions and avenues of possible future research will be set out in Chapter 6.

TABLE 3.2: **The different classification systems for objects in the dataset.**
A description of each of the ten original classes used by the human hand scanners is given [165], and their corresponding visual and main (real/not-real) classification is shown.

Original Class	Description	Visual Class	Real/Not-Real
Artefact	Residuals caused by problems in the image (e.g. diffraction spikes).	Artefact	Not-Real
Dipole	Residuals with roughly equal amounts of positive and negative flux, caused by errors in image registration.	Dipole/Saturated	Not-Real
Saturated Star	Residuals of stars that saturate the CCD.	Dipole/Saturated	Not-Real
Moving	Anything showing signs of motion between cutouts in different passbands.	Real	Real
Variable	Objects showing a record of long-term variability.	Real	Real
Transient	Objects with no observation history, no apparent host galaxy in the search image, and no motion.	Real	Real
SN Other	Objects that are thought to have a good chance of being SNe, but that don't fit nicely into any of the above classes.	Real	Real
SN Bronze	Point-like residuals at the centre of their host galaxies - most of the objects in this class later turn out to be either quasars (QSOs), active galactic nuclei (AGN) or foreground variable stars, and not SNe.	Real	Real
SN Silver	Point-like residuals having no apparent host galaxy - SNe much more luminous than their host galaxies usually fall into this class.	Real	Real
SN Gold	Possible SNe identified as point-like residuals that are not at the exact centre of their host galaxies.	Real	Real

Chapter 4

Feature Extraction



– http://xkcd.com/329/

4.1 Introduction

This chapter discusses the pre-processing applied to the data and the machine learning techniques employed in order to create our various feature sets for training and validation of our machine learning algorithms in Chapter 5. The pre-processing of our data is discussed in section 4.2, while the two machine learning techniques, namely Principal Component Analysis and Linear Discriminant Analysis (PCA and LDA) that we employed for feature extraction is discussed in section 4.3. A summary of our various feature sets are provided in section 4.4. It should be noted that the features for one object alone is referred to as a feature vector, while all the feature vectors for all the objects in a given data set is referred to as a feature set.

4.2 Pre-Processing

The raw data for our classification problem are the values of the pixels for the 51×51 pixel difference images. This means that, without any image processing or feature extraction, each object (consisting of a g, r and i-band image) in our data set is represented by 7803 values. However, because the objects in these images are centered, the outer pixels often carry very little distinguishing information compared to the inner pixels. Cutting out insignificant pixels can be beneficial to the classification process as it removes an excess of irrelevant data that can possibly confuse classifiers as well as swamp computational resources.

In order to determine which cropping size would work best for our data, we successively cropped the $N \times N$ pixel images in our training and validation sets to sizes of $N = 11$, $N = 21$, $N = 31$, $N = 41$ and $N = 51$, respectively. We then extracted our 28 different feature sets (see section 4.4) for each of these cases, and proceeded to train and validate our Support Vector Machine (SVM, see section 5.4.3) for each of the various feature sets in each of the five cases. It was found that the 31×31 pixel images gave the best results for the largest number of feature sets, with the additional benefit of being computationally less time-consuming than the original case of $N = 51$. Fig. 4.1 shows the g, i and r-band image of an object before and after cropping.

For thoroughness, and to ensure that the $N = 31$ case is also optimal for our remaining four classifiers, feature sets were built using both the cropped ($N = 31$) and the uncropped ($N = 51$) case. This is further discussed in section 4.4. Furthermore, we also tried to align images using rotations and translations - this did not lead to any improvement in classifier performance, however, and was therefore not implemented.

4.3 Feature Extraction

Machine learning methods need, as their input, features representing the particular instances. In principle, the pixel values of the images themselves could be used as features, but this might be impractical. The first issue here is a computational one: as each instance (object) in our data set consists of three 31×31 pixel images, the feature vector of one instance alone would comprise of 2883 features - a large number. Another issue is that we are working with highly compressible images - if we include all the pixel values of the cropped images we might confuse the classifiers with irrelevant features, as mentioned before. Having too large a feature space also puts us at risk of the curse of dimensionality.

To overcome these difficulties we employed two well-known machine learning algorithms, PCA and LDA, for the extraction of those features that represent the data most faithfully. Using PCA

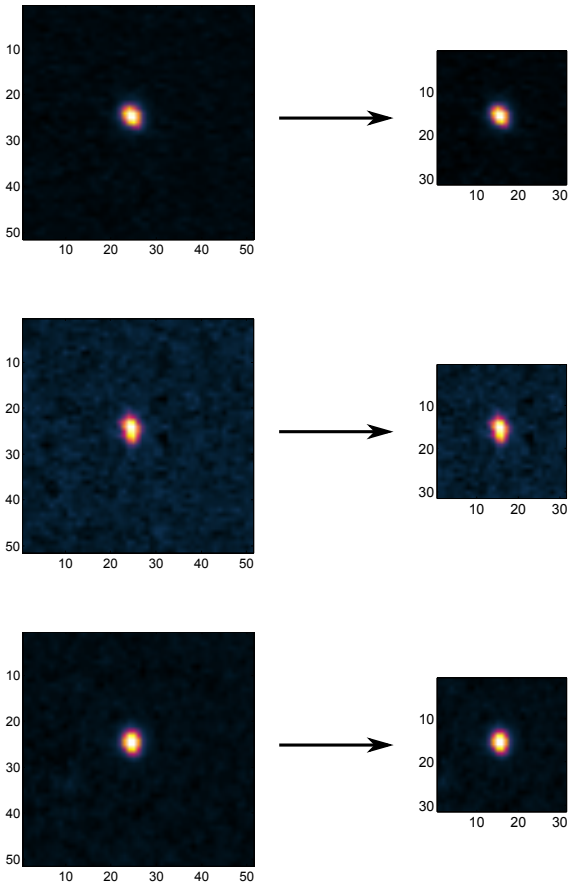


FIGURE 4.1: **Image cropping.**
The g (top), i (middle) and r- (bottom) band images of a supernova candidate being cropped from a size of 51×51 pixels to 31×31 pixels.

(section 4.3.1), we extracted features in two different ways. The first method, referred to as *Multi-class PCA*, is discussed in section 4.3.1.3 and comprises of carrying out PCA on the full training data set (see Fig. 3.7) while the second method, referred to as *Single-class PCA*, is discussed in section 4.3.1.4 and is based on carrying out PCA on each of the visual classes in the training set separately. LDA was carried out using the full training set, and is discussed in section 4.3.2.

4.3.1 Principal Component Analysis

This section details how PCA was used to extract features for our data. It discusses the specific algorithm we employed, the input data to the algorithm and the different ways in which we used PCA in order to derive our features, namely Multi-class PCA and Single-class PCA. The theoretical background to this section can be found in section 2.3.1.

4.3.1.1 The Algorithm

The algorithm we employed was taken from work done by Pedregosa et al. [166], who implemented the Probabilistic Principal Component Analysis (PPCA) model from the work of Tipping and Bishop [167].

As was seen in section 2.3.1, the M-dimensional feature vector $\mathbf{a}_n$ (consisting of PC weights) of a D-dimensional instance $\mathbf{x}_n$, where $M < D$, can be expressed as

$$\mathbf{a}_n = \mathbf{U}^{\mathrm{T}}(\mathbf{x}_n - \overline{\mathbf{x}}) \tag{4.1}$$

where $\overline{\mathbf{x}}$ is the average of the data vectors defined by equation 2.39 and $\mathbf{U}$ is a $D \times M$ dimensional matrix having the M largest eigenvectors (PCs) as columns. Fig. 4.2 illustrates the working of Eq. 4.1 and shows clearly the PC weights ($\mathbf{a}_n$) extracted as features.

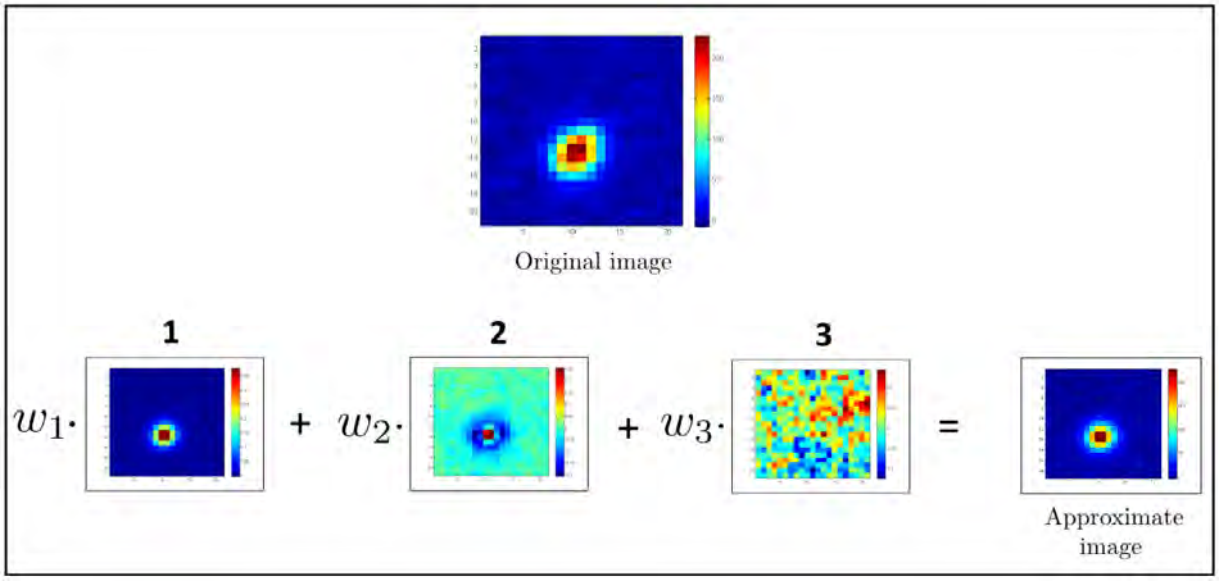


FIGURE 4.2: **PC weights.**
This figure shows the case where the three ($M = 3$) largest eigenvectors (each with a bold number at the top to show their rank in magnitude) are used to extract the feature vector $\mathbf{a}_n$ of an instance $\mathbf{x}_n$ (the "Original image" in the figure). The PC weights w_1, w_2 and w_3 necessary to arrive at an "Approximate image" as close as possible to $\mathbf{x}_n$ is calculated by Eq. 4.1 above and constitute the feature vector $\mathbf{a}_n$ of $\mathbf{x}_n$. In order to use these images in Eq. 4.1 they would of course first have to be expressed in vector format.

One disadvantage of the standard PCA formalism given in section 2.3.1 is the lack of a probability density model and likelihood measure. By deriving PCA from a density estimation perspective (PPCA) one permits its comparison to other density-estimation methods, creates the possibility of

statistical testing and allows the PCA model to be extended to a mix of such models. Using PPCA, $\mathbf{U}$ is determined by incorporating it with a set of hidden variables which is assumed to be Gaussian distributed and represent the weights to the directions corresponding to the columns of $\mathbf{U}$. Using this formulation, after marginalising over the hidden variables, we can obtain a maximum-likelihood estimator of $\mathbf{U}$, which is the best-estimated matrix of principal axes (rotated and scaled) of the data. For more detail regarding PPCA, see Chapter 12.2 in [92].

4.3.1.2 Creating the Input Vectors

In order to computationally obtain the PCs of a set of R training objects $I_1, I_2, \cdots, I_R$ with the algorithm mentioned above, it is first necessary to construct the input data matrix of these objects by representing each object I_n by its corresponding vector, Γ_n. This is done by expressing each of its g, i and r-band $N \times N$-pixel images as vectors, and then concatenating them. If I_{n-g}, I_{n-i} and I_{n-r} denotes the g, i and r-band images, respectively, of object I_n and are expressed as

$$I_{n-g} = \begin{bmatrix} g_{1,1} & \cdots & g_{1,N} \\ \vdots & \ddots & \vdots \\ g_{N,1} & \cdots & g_{N,N} \end{bmatrix} \qquad I_{n-i} = \begin{bmatrix} i_{1,1} & \cdots & i_{1,N} \\ \vdots & \ddots & \vdots \\ i_{N,1} & \cdots & i_{N,N} \end{bmatrix} \qquad I_{n-r} = \begin{bmatrix} r_{1,1} & \cdots & r_{1,N} \\ \vdots & \ddots & \vdots \\ r_{N,1} & \cdots & r_{N,N} \end{bmatrix} \tag{4.2}$$

then their corresponding vectors Γ_{n-g}, Γ_{n-i} and Γ_{n-r} are constructed as follows:

$$\Gamma_{n-g} = \begin{bmatrix} g_{1,1} \\ \vdots \\ g_{1,N} \\ \vdots \\ g_{2,N} \\ \vdots \\ g_{N,N} \end{bmatrix} \qquad \Gamma_{n-i} = \begin{bmatrix} i_{1,1} \\ \vdots \\ i_{1,N} \\ \vdots \\ i_{2,N} \\ \vdots \\ i_{N,N} \end{bmatrix} \qquad \Gamma_{n-r} = \begin{bmatrix} r_{1,1} \\ \vdots \\ r_{1,N} \\ \vdots \\ r_{2,N} \\ \vdots \\ r_{N,N} \end{bmatrix} . \tag{4.3}$$

The vectors Γ_{n-g}, Γ_{n-i} and Γ_{n-r} are then concatenated to form the objects representative vector Γ_n, as follows

$$\Gamma_n = \begin{bmatrix} \Gamma_{n-g} \\ \Gamma_{n-i} \\ \Gamma_{n-r} \end{bmatrix} = \begin{bmatrix} g_{1,1} \\ \vdots \\ g_{N,N} \\ i_{1,1} \\ \vdots \\ i_{N,N} \\ r_{1,1} \\ \vdots \\ r_{N,N} \end{bmatrix}. \tag{4.4}$$

The data matrix $\mathbf{X}$ for the set of R objects can then be constructed as

$$\mathbf{X} = \begin{bmatrix} \Gamma_1 & \Gamma_2 & \cdots & \Gamma_R \end{bmatrix}. \tag{4.5}$$

where the input to the algorithm is $\mathbf{X}^{\mathrm{T}}$.

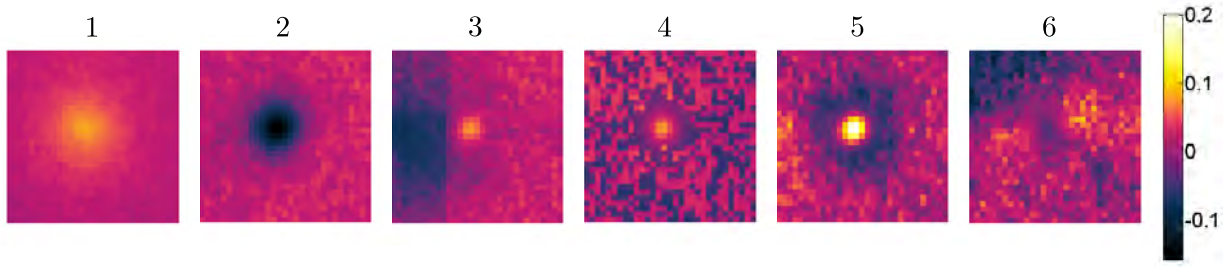
4.3.1.3 Multi-class PCA

As mentioned before, *Multi-class PCA* was carried out by doing PCA (with the algorithm discussed in section 4.3.1.1) on the full training data set mixing all classes together (see Fig. 3.7), whereafter a small fraction of the PCs were kept for the purpose of our feature extraction. To derive features, all objects (from both the training, validation and test sets) were expressed in vector form as shown in the previous section, and were then expressed as a linear combination of these PCs, with the coefficients (weights) being kept as the objects' features (as expressed in Eq. 4.1).

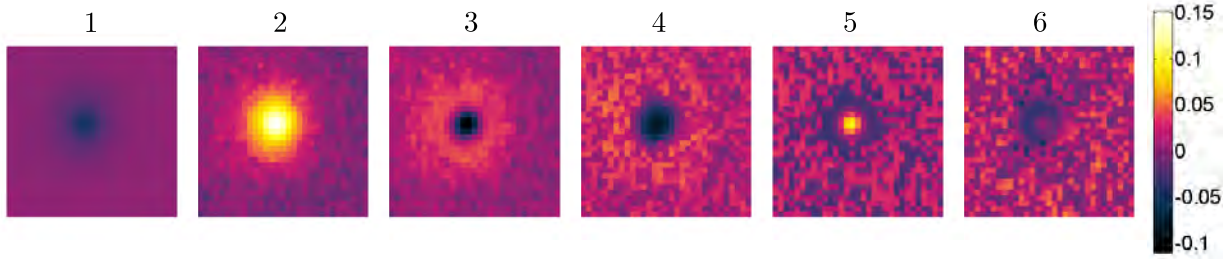
Fig. 4.3a shows the first six PCs of the full training set for the r-band images. It should be noted that the first PCs in the figure are less noisy than the last ones. The optimum number of PCs to use for feature extraction depends on the specific classifier and is a parameter that we optimised for each of our classifiers by training and validating them with feature sets derived by including either 0, 5, 10, 25, 50, 100 or 200 PCs, respectively (see section 4.4 for a discussion of our different feature sets).

4.3.1.4 Single-class PCA

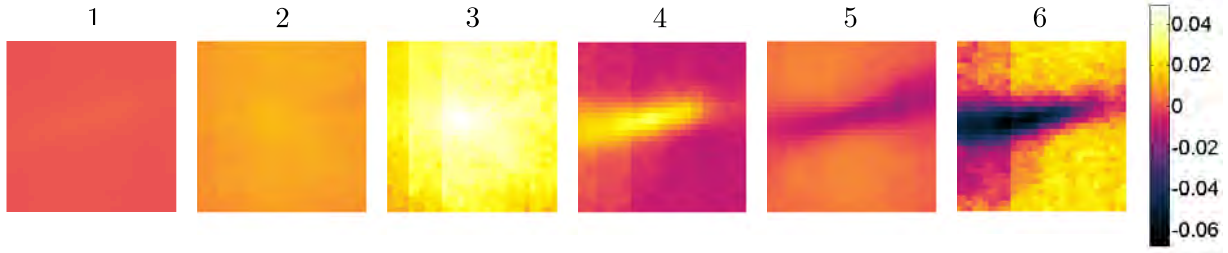
Another approach we used for feature extraction was to apply PCA separately to each visual class (see Table 3.2) in the training set, resulting in a unique set of PCs per visual class (see Fig. 4.3b,



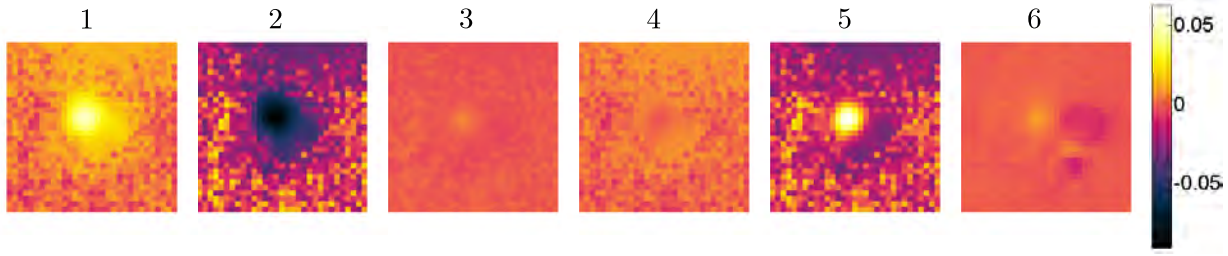
(A) Full training set.



(B) Real objects.



(C) Artefacts.



(D) Dipoles/saturated.

FIGURE 4.3: **Principal Component Analysis**
The first six PCA components in the r-band of (a) the full training set, (b) the training set objects belonging to the real class, (c) the training set objects belonging to the artefact class and (d) the training set objects belonging to the dipole/saturated class.

4.3c and 4.3d). To extract the features of an object I_n found in either the training, validation or test set, the object is first expressed in vector form and then reconstructed using the PCs of each visual class in turn, resulting in a reconstructed object associated with the real, artefact and saturated/dipole class denoted by $\widetilde{I}_{n(real)}$, $\widetilde{I}_{n(art)}$ and $\widetilde{I}_{n(sat/dip)}$ respectively. The error per pixel for the object, denoted by $\varepsilon_{n(class)}$ where $class \in \{real, art, sat/dip\}$, between each reconstructed object $\widetilde{I}_{n(class)}$ and the original object I_n is then calculated by taking the Euclidean distance between them and dividing by the number of pixels in an object, m, as is shown in Eq. 4.6.

$$\varepsilon_{n(class)} = \sqrt{\left(I_n - \widetilde{I}_{n(class)}\right)^2} \, /m \tag{4.6}$$

As each object consists of an $N \times N$ pixel g, i and r-band image, where $N = 31$, the number of pixels would be equivalent to $m = 3N^2$. It should also be noted that the first 15 PCs from each of the visual classes were used for the obtaining of the respective reconstructed objects. The three errors $\varepsilon_{n(real)}$, $\varepsilon_{n(art)}$ and $\varepsilon_{n(sat/dip)}$ calculated in this way are then used as features for object I_n.

As before, the algorithm for the calculation of the PCs was taken from the work done by Pedregosa et al. [166], and the data matrices for the individual visual classes were constructed in a similar manner as explained in section 4.3.1.2.

4.3.2 LDA

The LDA algorithm that we employed for the extraction of features for our data was taken from work done by Pedregosa et al. [166], who based their model on the work of Hastie et al. [95]. We discussed LDA in section 2.3.2 using the intuitive approach of Fisher's Linear Discriminant (FLD). The method used by Hastie, on the other hand, arrives at the exact same result obtained in section 2.3.2 by fitting a Gaussian density to the class data and assuming that the covariance matrices of all classes are identical. For more detail regarding this method, see Chapter 4.3 in Hastie's book [95].

LDA was carried out on the full training set. Seeing as objects are classified into one of two distinguishable classes (namely real and not-real), objects in the training, validation and test sets were therefore projected down to one dimension, resulting in only one LDA component being used as a feature for an object (see section 2.3.2). The input data matrix was constructed in a similar manner as explained in section 4.3.1.3, this time also including the classes of the objects (real or not-real).

Whether or not to include the LDA component in the feature vectors of objects depends on the specific classifier and is a parameter that we optimised for each of our classifiers by training and validating them with feature sets that both include and lack the component (see section 4.4).

4.4 Feature sets

After obtaining our features from PCA (namely the PC weights from Multi-class PCA and the reconstruction errors from Single-class PCA) and LDA, we constructed various feature sets with which to train, validate and test our classifiers - we did not want to select one feature set exclusively seeing as some classifiers might be more successful using one feature set than another, and this was still unclear at this stage. Varying the feature sets therefore served as a further method of optimisation for each classifier.

Exactly what the feature vectors in the feature sets comprised of and the manner in which a number of different feature sets were obtained is explained as follows:

- All feature sets included the three reconstruction errors obtained from Single-class PCA.
- The number of PC weights (obtained from Multi-class PCA) included in the sets were varied between 0, 5, 10, 25, 50, 100 and 200.
- Feature sets either included an LDA component or excluded it.
- Feature normalisation[1] was either carried out or omitted.
- For thoroughness, feature sets were either derived from cropped 31×31 pixel images or from uncropped images.

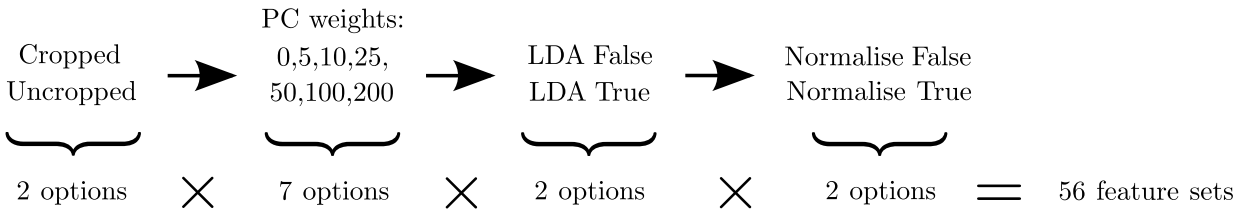


FIGURE 4.4: **Obtaining different feature sets.**
This diagram further illustrates the process of obtaining the 56 different feature sets following from the bullet-point discussion above. "LDA False" and "LDA True" correspond to excluding and including, respectively, an LDA component in the feature set, while "Normalise False" and "Normalise True" correspond to whether feature normalisation was omitted or carried out.

The above resulted in a grand total of 56 different feature sets to be used for training and validation purposes in order to optimise the different machine learning classifiers (see Fig. 4.4 for further clarification as to how 56 different feature sets were obtained). This is further discussed in Chapter 5. We also included the true classifications ("0" for not-real and "1" for real) and identity numbers of objects in the feature sets for training and identification purposes and for performance calculations.

[1]The feature data were normalised so that all features would have a standard deviation equal to 0.5 and a mean equal to zero. This was done for improving the efficiency with which various routines optimised the different learners' objective functions.

Chapter 5

Machine Learning Classification of Transient Survey Images

I visualize a time when we will be to robots what dogs are to humans, and I'm rooting for the machines.

– Claude Shannon

5.1 Introduction

This chapter discusses the training, validation, testing and performance results of our five machine learning algorithms for the problem of distinguishing between real and not-real objects. The performance measures used are discussed in section 5.2 while the training, validation and testing procedures are set out in section 5.3. The various learning methods we employed, as well as their testing and individual performance results, are described in section 5.4 - we used a Minimum Error Classifier (MEC, section 5.4.1), a Naïve Bayes (NB, section 5.4.2) classifier, a Support Vector Machine (SVM, section 5.4.3), a k-Nearest Neighbour (kNN, section 5.4.4) algorithm and an artificial neural network called SkyNet (section 5.4.5). The performance results of these classifiers are then compared, discussed and analysed in section 5.5.

5.2 Performance Measures

For optimisation, validation and testing of our learning algorithms we employed four performance measures often used in classification problems - the accuracy (A), precision (P), recall (R) and F_1-score. These measures are defined using the number of true positives (t_p), true negatives (t_n), false positives (f_p) and false negatives (f_n) as:

$$A = \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \tag{5.1}$$

$$P = \frac{t_p}{t_p + f_p} \tag{5.2}$$

$$R = \frac{t_p}{t_p + f_n} \tag{5.3}$$

$$F_1 = \frac{2PR}{P + R} \tag{5.4}$$

Precisely what these true/false positive/negative values constitute can easily be seen with the help of a confusion matrix (Table 5.1) where each column corresponds to instances in a predicted class, and each row corresponds to instances in an actual (true) class. With the help of Table 5.1 it is clear that true positives (t_p), for example, correspond to real objects that were correctly classified as real, and false positives (f_p) correspond to not-real objects incorrectly classified as real. False negatives (f_n) and true negatives (t_n) can be explained in a similar fashion. For the classification problem at hand, the positive class corresponds to our real objects, and the negative class corresponds to our not-real objects.

TABLE 5.1: **A confusion matrix.**
An archetype of a confusion matrix adjusted for our specific problem, where the positive class corresponds to real objects and the negative class to not-real objects. True/false positives/negatives are denoted as t_p, t_n, f_p and f_n respectively.

		Predicted Class	
		Real	Not-Real
True Class	Real	t_p	f_n
	Not-Real	f_p	t_n

Eq. 5.1 to 5.3 can be explained in a more intuitive way as follows: accuracy (A) is the fraction of all instances that was classified correctly, precision (P) is the fraction of positively classified objects that actually are positive, and recall (R) is the fraction of actual positive objects that was classified as positive by the classifier. The F_1-score (Eq. 5.4), with its worst value at zero and its best at 1, can be seen as the weighted average of R and P.

The specific classification problem at hand usually dictates which particular performance measures should be used for the measurement of classifier success. Accuracy, for example, can be misleading in problems where numbers of negative and positive instances are hugely different - a classifier always predicting "positive" will have a very good accuracy on a data set in which 99% of objects are in the positive class. Because our data consist of two classes (real and not-real) of roughly

similar sizes, the accuracy can be used as a sensible measure. A measure that is (in most cases) somewhat better than accuracy is the F_1-score - a measure of a classifier's accuracy punishing false positives and false negatives equally, but accounting for big class number hierarchies by weighting them with their inverse contribution to the whole data set.

In practical applications, we may be more concerned with false negatives than with false positives, and vice versa. For an example based on our problem, if the number of instances predicted to be real is small enough that humans can easily verify it, it would be preferable to minimise f_n and therefore maximise R. On the other hand, if we have too large a data stream making human checking infeasible, preventing the contamination of data might become important and in that case the better choice would be to minimise f_p and maximise P.

Because most learning algorithms output an instance's probability of being in a specific class, the trade-off between false negatives and false positives can be studied systematically. We do this by using a Receiver Operating Characteristic (ROC) curve - a plot displaying a binary classifier's performance as its discrimination threshold[1] is changed. A ROC curve is produced with the plotting of a classifier's True Positive Rate (TPR, equivalent to recall) versus its False Positive Rate (FPR) for different threshold values [168]. Eq. 5.5 shows how the FPR, also referred to as the fall-out, can be calculated.

$$FPR = \frac{f_p}{f_p + t_n} \tag{5.5}$$

A ROC statistic commonly employed for model comparison in the field of machine learning is the area under the curve (AUC or "Area Under Curve") - equivalent to the probability of a classifier correctly classifying an instance chosen at random [169]. As the name suggests, the AUC value for a given classifier is determined by integrating the classifier's ROC curve in order to yield the area under its curve.

For our real/not-real classification problem we chose accuracy and recall as our prime performance measures for the initial validation and optimisation phases of the classifiers. Because our two main classes (real and not-real) have roughly similar sizes, accuracy can be used as a good measure of success, while recall is an appropriate metric for our problem due to the fact that we are more worried about losing real objects (f_n's) than we are about the possible contamination of the set of classified real objects. It will be easy enough for humans to get rid of false positives afterwards. For the final testing stage where the withheld test data are used we quote all of the above measures (A, P, R, the F_1-score and the AUC statistic) and also supply the confusion matrix and the ROC

[1] If, for example, a binary classifier outputs the probability p of an instance being in a certain class, with the instance classified as part of the class if $0.5 \leq p \leq 1$ but classified as the other if $0 \leq p < 0.5$, the discrimination threshold is 0.5.

curve for each of the classifiers in order to provide a full picture of the results. Classifiers are then ordered from best- to worst-performing based on A, R and also the AUC value.

5.3 Training, Validation and Testing Procedures

This section discusses the procedures according to which our learning algorithms were trained, validated and tested using the different data sets described in Fig. 3.7.

In general, all our algorithms were initially trained and validated with all 56 different feature sets (see section 4.4). During this phase the most successful feature set for each classifier was determined and, where necessary, classifier parameters were optimised. The algorithms were then submitted to the final testing phase during which each classifier was tested with the feature set (now taken from the withheld test set) that proved most optimal for it during the initial validation phase mentioned above. The final testing phase of the algorithms yielded their respective final performance results for the real/not-real classification problem.

It should be noted that, even though some classifiers do not strictly require a validation phase, all algorithms were submitted to each of these phases in order to attain a sense of uniformity throughout the process (a fixed phase for feature set determination and a fixed phase for final testing), and because it allowed for continuous comparison between methods throughout all phases. Any exceptions to the general procedure given above will be discussed in the sections detailing the testing of the individual algorithms themselves (see section 5.4).

5.4 Machine Learning Algorithms and their Implementations

This section discusses the implementation, training, validation and testing of each of the five algorithms that we employed for our classification problem, and gives the performance results obtained during final testing for each. It should be noted that even though some results are given here, they are only discussed and compared in section 5.5. Detail on the theoretical working of these algorithms can be found in Chapter 2.

5.4.1 Minimum Error Classification

Minimum Error Classification (MEC) is our most basic classifier. It does not really constitute a machine learning algorithm, and was merely used in order to see what the classification potential of the simplest of classifiers would be and to create a baseline for comparison. We implemented MEC in MATLAB® [170] on a simple laptop computer (Intel Core i7, with a clock speed of 2.4 GHz).

5.4.1.1 The Algorithm

MEC takes as its inputs only the reconstruction error features derived using Single-class PCA (see section 4.3.1.4). It first assigns an input instance $\mathbf{x}_n$ to the visual class ($\mathcal{C}_{\text{visual}}$) associated with its minimum reconstruction error as

$$\mathcal{C}_{\text{visual}} = \underset{class}{\text{argmin}} \left(\varepsilon_{n(class)}\right) \tag{5.6}$$

where $class \in \{real, art, sat/dip\}$ and $\varepsilon_{n(class)}$ is the reconstruction error calculated as shown in section 4.3.1.4. This intuitively reflects the logical reasoning that, on average, an instance should belong to the visual class whose PCA reconstruction of it results in the smallest error.

Finally, if the instance's minimum error corresponds to either the artefact or the dipoles/saturated visual class, it is assigned to the not-real main class, whereas if the minimum error corresponds to the real visual class, the instance will be assigned to the real main class. For a reminder on how the different classification schemes work, see Table 3.2.

5.4.1.2 Testing

MEC is not a machine learning algorithm in the full sense and as such required no training and was directly tested on the withheld test set. Due to the fact that MEC only takes the reconstruction errors as inputs, only 4 of the 56 feature sets mentioned in section 4.4 could be tested on the classifier. The best results were obtained by using a non-normalised feature set derived from cropped 31×31 pixel images, yielding an accuracy of 84%, a recall of 92%, a precision of 83% and an F_1-score of 87%. The corresponding confusion matrix can be seen in Table 5.2.

TABLE 5.2: **The confusion matrix for MEC.**

	Real	**Not-Real**
Real	3559	324
Not-Real	754	2237

5.4.2 Naïve Bayes

A Naïve Bayes (NB) classifier is a supervised machine learning algorithm, and is used here for our binary classification problem of distinguishing between real and not-real objects. The reader is reminded that the theoretical background required for this section is found in section 2.2.1. I

designed the NB algorithm and implemented it in MATLAB® on a simple laptop computer (Intel Core i7, with a clock speed of 2.4 GHz) - see Appendix D for the code used.

5.4.2.1 The Algorithm

As mentioned in section 2.2.1.2, the NB classifier's approach to the classification of an input instance $\mathbf{x}$ is given by

$$\mathcal{C}_{NB} = \underset{\mathcal{C}_m}{\operatorname{argmax}}\, P(\mathcal{C}_m) \prod_i P(x_i|\mathcal{C}_m) \tag{5.7}$$

where $P(\mathcal{C}_m)$ is the prior probability of the class $\mathcal{C}_m$, $P(x_i|\mathcal{C}_m)$ is the conditional probability of x_i given class $\mathcal{C}_m$, where x_i is value of the i'th feature of the input instance $\mathbf{x}$, and $\mathcal{C}_{NB}$ is the class output of the learning algorithm. $\mathcal{C}_{NB}$ corresponds to the class $\mathcal{C}_m$ with the highest probability of being correct, given the feature values $(x_1, x_2, \ldots, x_N)$ describing the unclassified input instance $\mathbf{x}$.

The training phase of the NB classifier consists of estimating the $P(\mathcal{C}_m)$ and $P(x_i|\mathcal{C}_m)$ terms - this is done by counting their respective frequencies in the training data. Once training is completed, the algorithm can be used for the classification of new data instances by using the approach in Eq. 5.7. It should be noted that in our case, $\mathcal{C}_m \in \{\mathcal{C}_{\text{real}}, \mathcal{C}_{\text{not-real}}\}$. Using Eq. 5.7, we therefore essentially calculate (for a given input instance $\mathbf{x}$) the probabilities $P(\mathcal{C}_{\text{real}}|\mathbf{x})$ and $P(\mathcal{C}_{\text{not-real}}|\mathbf{x})$, and assign $\mathbf{x}$ to the class for which this probability is a maximum.

The $P(\mathcal{C}_m)$ terms can be very easily estimated merely by determining the fraction of training data instances belonging to the class $\mathcal{C}_m$. The $P(x_i|C_m)$ terms, on the other hand, are estimated by binning the training instances belonging to a given class $\mathcal{C}_m$ according to each feature, respectively, where the number of equally spaced bins for each feature is calculated such that there is an average of four training data points in each bin.

The probability $P(x_i|C_m)$ for a new unclassified instance $\mathbf{x}$ is then estimated by noting which fraction of the training instances in class $\mathcal{C}_m$ lies in the bin associated with the value x_i of the i'th feature of $\mathbf{x}$.

5.4.2.2 Testing

The NB classifier was trained and validated (using the data sets shown in Fig. 3.7) with each of the 56 feature sets mentioned in section 4.4. Upon completion of the validation phase, it was found that the best results came from using a non-normalised feature set derived from cropped 31×31 pixel images consisting of three reconstruction errors and 50 PC weights. When tested on

the withheld test data, this model yielded an accuracy of 77%, a recall of 86%, a precision of 77% and an F_1-score of 81%. The corresponding confusion matrix is shown in Table 5.3.

TABLE 5.3: **The confusion matrix for NB.**

	Real	**Not-Real**
Real	3333	550
Not-Real	998	1993

5.4.3 Support Vector Machine

A Support Vector Machine (SVM) is a supervised machine learning algorithm, and is used here for our binary classification problem of distinguishing between real and not-real objects. The reader is reminded that the theoretical background required for this section is found in section 2.2.3. The implementation of the SVM algorithm was done with scikit-learn [166] (version 0.16.0), an open source machine learning library for Python[2] (we used Python 2.7.9), on a simple desktop computer (with a clock speed of 2.4 GHz).

5.4.3.1 The Algorithm

The algorithm we employed was taken from the work done by Pedregosa et al. [166] - we implemented their `SVC` class, which, equivalent to what was explained in section 2.2.3, solves the primal problem of minimising

$$C\sum_{n=1}^{N}\xi_n + \frac{1}{2}||\mathbf{w}||^2 \tag{5.8}$$

with respect to $\mathbf{w}$, b and $\{\xi_n\}$, where $\mathbf{w}$ is the weight vector, b is the bias, ξ_n is the slack variable of a training instance $\mathbf{x}_n$ and C is a regularisation parameter, referred to here as the soft margin parameter. The above minimisation has to be carried out subject to the constraints

$$t_n y(\mathbf{x}_n) - 1 + \xi_n \geq 0 \tag{5.9}$$

$$\xi_n \geq 0, \tag{5.10}$$

where $t_n \in \{-1, 1\}$ is the target value of $\mathbf{x}_n$ and $y(\mathbf{x}_n)$ is the target function having the form

[2] https://www.python.org/

$$y(\mathbf{x}_n) = \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) + b \tag{5.11}$$

where $\phi(\mathbf{x}_n)$ is a feature-space transformation. For more detail regarding the above equations, their parameters and their interpretations, see section 2.2.3 - the above equations are equivalent to Eq. 2.23, A.23, A.26 and 2.15, respectively.

It is necessary to specify the kernel to be used in the implementation, and subsequently our SVM makes use of the Gaussian kernel, also referred to as the Radial Basis Function (RBF) kernel. The kernel is expressed as follows

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma||\mathbf{x} - \mathbf{x}'||^2), \tag{5.12}$$

where $\gamma = 1/2\sigma^2$, with σ being the Gaussian's width and $||\mathbf{x} - \mathbf{x}'||^2$ is the squared Euclidean distance between $\mathbf{x}$ an $\mathbf{x}'$. This kernel's associated feature vector is of infinite dimensionality.

For an SVM using a Gaussian kernel, there are two parameters the user has to supply that need optimisation: C and γ. The soft margin parameter C manages the trade-off between training instance misclassification and decision surface simplicity, with a low C resulting in a smooth decision boundary and a high C in greater classification accuracy. The γ parameter, on the other hand, defines the reach of a training instance's influence, with a low γ indicating a large influence and a high value indicating the opposite. An appropriate choice of values for these two parameters are critical to the performance of the SVM - their optimisation is discussed in section 5.4.3.2.

For the case of binary classification it is possible to employ Platt scaling [105] in order to output the probability of an instance being in a certain class (the full theoretical background to Platt scaling can be seen in section 2.2.3.4, and should be referred to in the case of confusion). This involves the fitting of a logistic sigmoid to an already trained SVM's outputs, where the sigmoid's parameters (note Eq. 2.26) can be determined through the use of a maximum likelihood function optimised, in our case, on the same training set that the SVM was trained with. To prevent over-fitting, the algorithm also carries out a further cross-validation on the training set.

For our problem, we employ Platt scaling for the purpose of outputting $P(\text{real})$ for every input instance, such that it will be classified as a real object when $0.5 \leq P(\text{real}) \leq 1$ and as a not-real object when $0 \leq P(\text{real}) < 0.5$.

5.4.3.2 Testing

The training phase (using the training set shown in Fig. 3.7) of the SVM was used for the optimisation of the C and γ parameters. For each of the 56 feature sets mentioned in section 4.4, C and γ were optimised for by using a grid search and T-fold cross-validation (see section 2.2.3.4 for detail on cross-validation), where $T = 3$ was used to split the training set into three equally-sized groups. The values of C and γ that were explored with the grid search ranged (for both parameters) from 10^{-3} to 10^{3} in powers of 10 - this range is suggested by Pedregosa et al. [166]. For each feature set, cross-validation was therefore carried out for each parameter pair in the grid, yielding an averaged accuracy score for each pair, after which the most optimal C-γ pair could then be selected.

After each feature set's optimal C and γ value have been determined using the training data, each of the 56 models were then validated using their corresponding validation feature sets. It was found that the optimal results came from $C = 1000$, $\gamma = 0.1$ and a normalised feature set derived from cropped 31×31 pixel images consisting of the three reconstruction errors, 100 PC weights and an LDA component. When tested on the withheld test data, this model yielded an accuracy of 86%, a recall of 90%, a precision of 85% and an F_1-score of 87%. The corresponding confusion matrix is given in Table 5.4.

TABLE 5.4: **The confusion matrix for SVM.**

	Real	**Not-Real**
Real	3514	369
Not-Real	605	2386

5.4.4 k-Nearest Neighbours

This section discusses the use of a k-Nearest Neighbours (kNN) algorithm for our supervised binary classification problem of distinguishing between real and not-real objects. The reader is reminded that the theoretical background required for this section is found in section 2.2.2. As for SVM, the implementation of the KNN algorithm was done with scikit-learn [166] (version 0.16.0), an open source machine learning library for Python (we used Python 2.7.9), on a simple desktop computer (with a clock speed of 2.4 GHz).

5.4.4.1 The Algorithm

The algorithm we employed was taken from the work done by Pedregosa et al. [166] - we used their `KNeighborsClassifier` method which, as also explained in section 2.2.2, classifies each input query instance based on a majority class vote among its k nearest neighbours, where the integer value k is user-specified. We optimised for the value of k, as is discussed in section 5.4.4.2. We mentioned in Chapter 2 that one can implement weights for the k nearest neighbours so that the neighbours closer to the query instance contribute more to the vote - we chose not to do this, and so our weights are uniform.

5.4.4.2 Testing

The kNN algorithm was trained (using the training set shown in Fig. 3.7) with each of the 56 feature sets mentioned in section 4.4, yielding 56 different models. For each of these models, the optimal value for k was selected by validating a given model using each of the following values for k: 5, 10, 15, 20 and 25. Upon completion of the validation phase, it was found that the optimal results came from using $k = 10$ and a normalised feature set derived from cropped 31×31 pixel images consisting of three reconstruction errors, 50 PC weights and no LDA component. When tested on the withheld test data, this model yielded an accuracy of 89%, a recall of 90%, a precision of 91% and an F_1-score of 90%. The corresponding confusion matrix is shown in Table 5.5.

TABLE 5.5: **The confusion matrix for kNN.**

	Real	**Not-Real**
Real	3506	377
Not-Real	363	2628

5.4.5 Artificial Neural Network - SkyNet

This section discusses the use of an Artificial Neural Network (ANN) for our supervised binary classification problem of distinguishing between real and not-real objects. The reader is reminded that the theoretical background required for this section is found in section 2.2.4. The University College London[3] (UCL) Physics and Astronomy Department's Splinter computer cluster (Intel, with a clock speed of 2.4 GHz - we used only one core) was used for the implementation of the algorithm.

[3] http://www.ucl.ac.uk/

5.4.5.1 The Algorithm

We employed SkyNet (version 1.1), a well-known ANN training tool created by Graff et al. [88], as our neural network algorithm. SkyNet can train feed-forward neural networks applicable to a wide range of machine learning problems - examples include classification, regression, clustering, dimensionality reduction and density estimation. It also gives the user the option of training an autoencoder (see section 2.2.4.3), a network that maps its inputs back to themselves - such networks are often used for providing a more intuitive way of carrying out non-linear dimensionality reduction. SkyNet further allows the user to train a Recurrent Neural Network (RNN) - a network in which the links between nodal units form a directed cycle in which nodes have feedback connections, allowing the network to display dynamic temporal behaviour. These networks can employ their internal memory for the processing of arbitrary input sequences, and are therefore often used in this capacity.

SkyNet provides the option of using a pre-training algorithm for the obtaining of network parameters (weights and biases) close to the training objective function's global optimum - an alternative to using an arbitrary initial state from which the training process is started, and a method that can therefore possibly ease the process of training complex networks. Furthermore, SkyNet uses convergence criteria in order to prevent over-fitting. More information on various variants of methods employed by SkyNet for more efficient optimisation of networks can be found in the paper by Graff et al. [88].

It was mentioned in section 2.2.4.2 that a universal approximation theorem [106] states that any continuous function can be approximated to a given accuracy with a neural network having three or more nodal layers, guaranteed the activation functions are not polynomials, are piecewise continuous and are locally bounded [88]. SkyNet uses the logistic sigmoid function (see Eq. 5.13) as the activation function for its hidden layers and therefore satisfies these conditions. We therefore decided to implement a 3-layer feed-forward neural network consisting of an input, hidden and output nodal layer.

$$g^{(1)}(x) = \frac{1}{1 + e^{-x}}. \tag{5.13}$$

Section 2.2.4.2 further mentions that one hidden layer (as said above) having $2N + 1$ nodes (N being the amount of inputs) is the optimal neural network structure for the approximation of continuous functions without the algorithm doing unnecessary work [107–109]. We decided to follow these guidelines for choosing the number of hidden nodes in our networks as long as it was computationally feasible - in the case of our largest feature set, for example, $N = 204$, resulting in a very infeasible and large network structure should the $2N + 1$ rule be followed strictly.

The activation function for SkyNet's output nodes are shown in Eq. 5.14. SkyNet provides continuous outputs for classification problems, interpreted as the probabilities of an input instance belonging to a specific class. To achieve this, the softmax function (see Eq. B.5) is applied to the output values so that they sum to unity and are non-negative. For our classification problem, two output nodes are used (one for each of our main classes) so that the final outputs they yield, given an input instance $\mathbf{x}_n$, are equivalent to $P(\mathcal{C}_{\text{real}}|\mathbf{x}_n)$ and $P(\mathcal{C}_{\text{not-real}}|\mathbf{x}_n)$. The instance $\mathbf{x}_n$ is then assigned to the class with the maximum probability.

$$g^{(2)}(x) = x. \tag{5.14}$$

In summary, then, we employed a 3-layer feed-forward neural network consisting of an input layer having N nodes (where N is the number of input features), a hidden layer with $2N + 1$ nodes (where feasible) and an output layer consisting of 2 nodes (equivalent to the amount of classes we are trying to distinguish between). Upon running the SkyNet algorithm, the user has to supply information regarding different possible settings: whether the pre-training method should be used, whether or not an autoencoder or RNN should be implemented, whether the input data should be whitened[4] and which parameters to use for the determination of convergence, to name the important ones. Details on how we chose these settings can be found in section 5.4.5.2. Of course, the number of network layers and the nodes in each layer should also be specified by the user.

5.4.5.2 Testing

Preliminary tests using the training and validation data sets (see Fig. 3.7) and a fixed feature set were carried out in order to establish whether our network should be configured as either an RNN or an autoencoder, and whether the pre-training method should be employed. The network was tested once in its standard configuration, with all three these options disabled, after which it was tested once for each of the options enabled in turn. It was found that each of these options, once enabled, did not yield any notable changes in network performance for our specific problem, and it was on this basis that they were thus deemed unnecessary. In fact, enabling the options only served to lengthen the algorithm's computational running time. It was furthermore decided that the input features to the network would be left unwhitened, seeing as our own feature sets already incorporated such measures, and that the average error-squared of outputs from the network would be used as the measure that SkyNet employs for determining convergence of the network. For more information about these settings, see the paper [88].

After the above-mentioned settings were determined, we trained and validated the neural network algorithm with each of the 56 feature sets mentioned in section 4.4, adjusting each network's input

[4] In SkyNet's context, the normalisation of input data to the network is called whitening.

and hidden nodes in accordance to the number of input features a given feature set had. This resulted in 56 different neural network models. Upon completion of the training and validation phase, it was found that the optimal results came from using a non-normalised feature set derived from cropped 31×31 pixel images consisting of three reconstruction errors, 200 PC weights and no LDA component. When this model was tested on the withheld test data, it yielded an accuracy of 88%, a recall of 89%, a precision of 90% and an F_1-score of 89%. The corresponding confusion matrix is shown in Table 5.6.

For the testing and validation of networks using feature sets with a smaller number of features, the $2N+1$ rule mentioned before was used for determining the number of hidden nodes in the network. For bigger feature sets (like the optimal feature set determined above), though, that would yield a number of hidden nodes too large to be computationally feasible. For the optimal feature set above, we used 100 hidden nodes (instead of 407), and the algorithm converged after 140 iterations.

TABLE 5.6: **The confusion matrix for SkyNet.**

	Real	**Not-Real**
Real	3461	422
Not-Real	399	2592

5.5 Results and Analysis

This section summarises, analyses, adds to and compares the classification results obtained in section 5.4. Section 5.5.1 provides the ROC curve for each classifier and discusses the best-performing classifier, section 5.5.2 discusses inter-classifier agreement and section 5.5.3 describes the classifiers' performance on the different visual classes. Incorrectly classified objects are studied in section 5.5.4, while section 5.5.5 compares our results with the human hand scanners mentioned in Chapter 3. Finally, section 5.5.6 discusses the performance of our classifiers on spectroscopically confirmed SNe. It should be noted that the results presented here appear in the paper by L. du Buisson et al. [1].

5.5.1 Best-Performing Classifier

This section discusses the ROC curves and "Area Under the Curve" (AUC) values obtained for our five classifiers, after which it compares their performance results and notes the best-performing learning algorithm.

As was mentioned in section 5.2, one needs probabilistic outputs with a well-defined discrimination threshold from a classifier in order to obtain its ROC curve. Our SVM, for example, outputs $P(\text{real})$

for a given input instance, such that it will be classified as a real object when $0.5 \leq P(\text{real}) \leq 1$ and as a not-real object when $0 \leq P(\text{real}) < 0.5$, corresponding to a discrimination threshold of 0.5. SkyNet outputs two probabilities given an input instance: $P(\text{real})$ and $P(\text{not-real})$. These probabilities sum to one, and so the decision as to the classification of an instance is similar to that of the SVM, also with a discrimination threshold of 0.5.

Our other classifiers (MEC, NB and kNN), on the other hand, do not provide us with such convenient outputs: MEC assigns an instance to the class corresponding to the minimum reconstruction error, kNN is based on a majority class vote among nearest neighbours, and the probability-outputs of NB do not sum to one. It is therefore necessary to convert the outputs of these classifiers into measures that can be used in a similar fashion as probabilities could, with well-defined discrimination thresholds.

In the case of MEC, an object is classified based on which class (real or not-real) the minimum reconstruction error corresponds to (as explained in Section 5.4.1). There are three error-values - one for each of the visual classes. If the minimum error corresponds to that of the "real" visual class, the object is classified as real, while if the minimum error corresponds to either the "artefact" or the "dipoles/saturated" visual class, the object is classified as being not-real. For our goal here, we first note that only the error-value corresponding to the minimum error value of the two visual classes ("artefact" and "dipoles/saturated") effectively matters when deciding whether an object is real or not-real - leaving us with two error-values to work with: one representing the real class and one representing the not-real class. We then create an estimator of the probability of an object being real, $\widehat{P}(\text{real})$, as

$$\widehat{P}(\text{real}) = 1 - \frac{\varepsilon_{real}}{\varepsilon_{real} + \varepsilon_{not-real}}, \tag{5.15}$$

where ε_{real} denotes the error-value corresponding to the real main class, and $\varepsilon_{not-real}$ is the minimum error-value of the visual classes corresponding to the not-real main class, as explained above. In the default classification case (discussed in Section 5.4.1) where an instance is classified solely based on the minimum error-value, an object will here be classified as real if $0.5 \leq \widehat{P}(\text{real}) \leq 1$ and as a not-real object if $0 \leq \widehat{P}(\text{real}) < 0.5$, corresponding to a discrimination threshold of 0.5. This discrimination threshold can now be varied from 0 to 1 in order to construct the ROC curve for MEC.

The NB classifier outputs two probability values (that do not sum to one) for our problem: the probability of an input instance belonging to the real class, denoted here as $P(\text{real})$, and the probability of it belonging to the not-real class, denoted here as $P(\text{not-real})$. The instance is assigned to the class yielding the highest of these probabilities. For the goal of converting this scheme to one where we have a well-defined discrimination threshold, we employ a transformation

similar to the one in Eq. 5.15 used for MEC. We rescale the probabilities in such a way that the estimated probability of an object being real, $\widehat{P}(\text{real})$, is now

$$\widehat{P}(\text{real}) = \frac{P(\text{real})}{P(\text{real}) + P(\text{not-real})}. \tag{5.16}$$

In the default classification case (discussed in section 5.4.2) where an instance is classified solely based on the class yielding the highest probability P, an object will here be classified as real if $0.5 \leq \widehat{P}(\text{real}) \leq 1$ and as a not-real object if $0 \leq \widehat{P}(\text{real}) < 0.5$, corresponding to a discrimination threshold of 0.5. This threshold can now be varied for the purposes of constructing a ROC curve.

In the case of kNN, an instance is classified based on the majority class vote of its k nearest neighbours. This can very simply be adjusted to our goals merely by introducing an estimator of the probability of an object being real, $\widehat{P}(\text{real})$, as

$$\widehat{P}(\text{real}) = \frac{k_{real}}{k_{real} + k_{not-real}}, \tag{5.17}$$

where k_{real} is the number of real nearest neighbours, $k_{not-real}$ is the number of not-real nearest neighbours, and $k_{real} + k_{not-real} = k$. In the default classification case (discussed in section 5.4.4) where an instance is classified solely based on a majority class vote, an object will here be classified as real if $0.5 \leq \widehat{P}(\text{real}) \leq 1$ and as a not-real object if $0 \leq \widehat{P}(\text{real}) < 0.5$, corresponding to a discrimination threshold of 0.5. This threshold can now be varied for the construction of a ROC curve.

The ROC curves for each of our five classifiers were constructed by using the outputs each of them obtained on the test data and varying their respective discrimination thresholds between 0 and 1 in order to see how their performance with regard to the TPR and FPR changed. The AUC statistic (see section 5.2) for each classifier was also calculated - the ROC curves for the classifiers as well as their respective AUC values can be seen in Fig. 5.1.

Table 5.7 summarises the classification results of the various classifiers on the test set (most of these results were also given in section 5.4), and orders them from best-performing to worst-performing with respect to recall, accuracy and the AUC value. This table, along with Fig. 5.1, represents some of the main results of this thesis. It is interesting that kNN, a very basic nonparametric learner, yielded the best results for all performance measures apart from recall, followed closely by SVM and SkyNet. Interestingly, our simplest classifier, MEC, yielded the best results for the recall measure. For small FPR values, though, Fig. 5.1 shows that its TPR lagged significantly. The differences in results between the best-performing classifiers (kNN, SkyNet and SVM) are very small and better optimisation of these learning algorithms could almost certainly result in a change

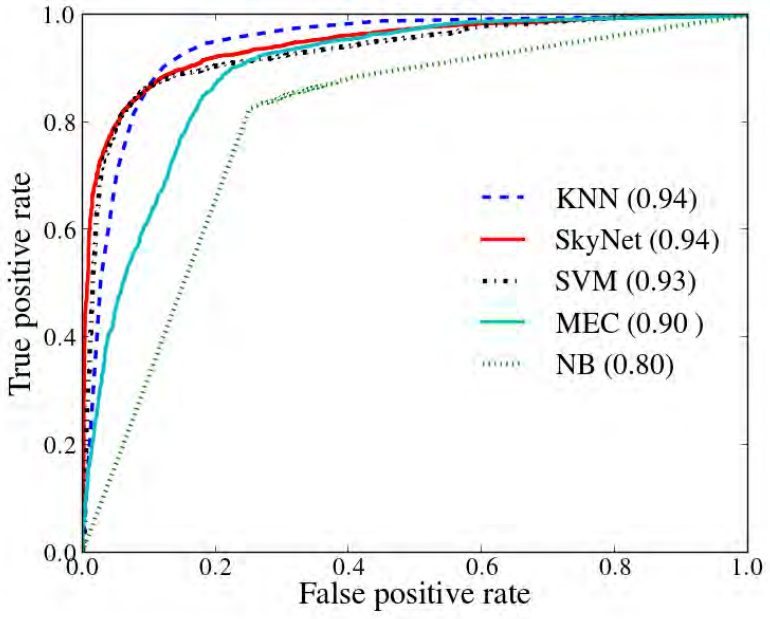


FIGURE 5.1: **ROC curve for the various classifiers.**
The ROC curve and Area Under Curve (AUC) for each of the five classifiers. The AUC is indicated in brackets next to each classifiers name, and classifiers are listed in order from best-performing to worst-performing, based on the AUC statistic. It should be noted that changing the threshold value of kNN, for example, can easily result in a recall (true positive rate) of 96% with only a slight penalty in the false positive rate - an encouraging sign.

in the final ordering obtained and shown in Table 5.7. From the viewpoint of substituting human hand scanners, the fact that multiple algorithms performed this well is very encouraging.

5.5.2 Inter-Classifier Agreement

For an idea of the inter-classifier agreement with regard to incorrectly and correctly classified objects, we make use of the Cohen's Kappa coefficient κ [172]. This statistical measure of the agreement between two classifiers is thought to be more accurate than a basic percent agreement calculation, seeing as it also takes chance agreement into account. For two classifiers each classifying N instances into one of K mutually exclusive classes, the Cohen's Kappa coefficient κ is determined as

$$\kappa = \frac{P(a) - P(b)}{1 - P(b)}. \tag{5.18}$$

Here, $P(a)$ denotes the observed relative agreement while $P(b)$ denotes the hypothetical probability of chance agreement. To illustrate how $P(a)$ and $P(b)$ are calculated, consider the following

TABLE 5.7: **Results table.**
A summary of the performance results of the various classifiers on the validation data, ordered from best-performing (top) to worst-performing (bottom) based on accuracy (A), recall (R) and the AUC value. The best result for each performance metric is indicated in bold red. The true labels are on the vertical side of the confusion matrices, and the predicted labels are on the horizontal side (see Table 5.1).

ML Technique	AUC	A	R	P	F1	Confusion Matrix		
k-Nearest Neighbours (kNN) Section 5.4.4	**0.94**	**0.89**	0.90	**0.91**	**0.90**		**Real**	**Not-Real**
						Real	3506	377
						Not-Real	363	2628
SkyNet Section 5.4.5	**0.94**	0.88	0.89	0.90	0.89		**Real**	**Not-Real**
						Real	3461	422
						Not-Real	399	2592
Support Vector Machine (SVM) Section 5.4.3	0.93	0.86	0.90	0.85	0.87		**Real**	**Not-Real**
						Real	3514	369
						Not-Real	605	2386
Minimum Error Classification (MEC) Section 5.4.1	0.90	0.84	**0.92**	0.83	0.87		**Real**	**Not-Real**
						Real	3559	324
						Not-Real	754	2237
Naïve Bayes (NB) Section 5.4.2	0.80	0.77	0.86	0.77	0.81		**Real**	**Not-Real**
						Real	3333	550
						Not-Real	998	1993

example: two classifiers each having to classify each of the objects in the same set as being either real or not-real. This scenario is depicted in Table 5.8, where A and B denote the classifiers, data on the diagonal slanting left shows the count of agreements and the data on the diagonal slanting right shows the count of the disagreements between the classifiers. Here we have $N = 100$ and $K = 2$.

TABLE 5.8: **Example classification results.**
An example of the classification results of two fictional classifiers, A and B, that both classified each one of 100 objects into one of two classes (real or not-real).

		A	
		Real	Not-Real
B	Real	30	20
	Not-Real	10	40

In Table 5.8, note that there were 40 objects classified as not-real by both classifier A and B, and 30 objects that were denoted as real by both classifiers. The observed relative agreement $P(a)$ is then calculated as

$$P(a) = \frac{40 + 30}{100} = 0.7. \tag{5.19}$$

To calculate $P(b)$, we note that classifier A classified 60 objects as not-real and 40 objects as real - in other words, in 60% of cases it classified an object as not-real, and in 40% of cases it classified an object as real. In 50% of cases, B classified objects as not-real. The probability of both A and B classifying an object as not-real is therefore $0.6 \times 0.5 = 0.3$, while the probability of both classifying an object as real is $0.4 \times 0.5 = 0.2$. The overall probability of chance agreement is therefore

$$P(b) = 0.3 + 0.2 = 0.5. \tag{5.20}$$

Applying Eq. 5.18 for Cohen's Kappa κ we then find

$$\kappa = \frac{P(a) - P(b)}{1 - P(b)} = \frac{0.7 - 0.5}{1 - 0.5} = 0.4. \tag{5.21}$$

If the two classifiers are in perfect agreement, $\kappa = 1$, whereas $\kappa = 0$ when classifiers have no agreement between them apart from that which is expected by chance alone. There are different opinions as to the precise interpretation of the value of κ (see, for example, the guidelines by Fleiss et al. [173], Altman [174] and Landis & Koch [175], respectively), but a rough idea of such interpretations can be found in Table 5.9, as put forth by Landis & Koch [175].

TABLE 5.9: **Interpretation of the Cohen's Kappa value.**
Guidelines to the strength of agreement between two classifiers based on the value of the Cohen's Kappa coefficient κ, as outlined by Landis & Koch [175].

Kappa Statistic	Strength of Agreement
$\kappa \leq 0.00$	None
$0.00 < \kappa \leq 0.20$	Slight
$0.20 < \kappa \leq 0.40$	Fair
$0.40 < \kappa \leq 0.60$	Moderate
$0.60 < \kappa \leq 0.80$	Substantial
$0.80 < \kappa < 1.00$	Almost Perfect
$\kappa = 1.00$	Perfect

Fig. 5.2 gives the κ-values for all pairs of our various classifiers (MEC, NB, SkyNet, kNN and SVM) for their performance during final testing. From Fig. 5.2 it can be seen that NB and SVM, and NB and MEC, have a stronger agreement ("moderate") between them than any of the other pairs of classifiers have. The fact that none of the best-performing classifiers are in strong agreement is a good indication that combining different classifiers' predictions in an ensemble classifier might yield improved classification results.

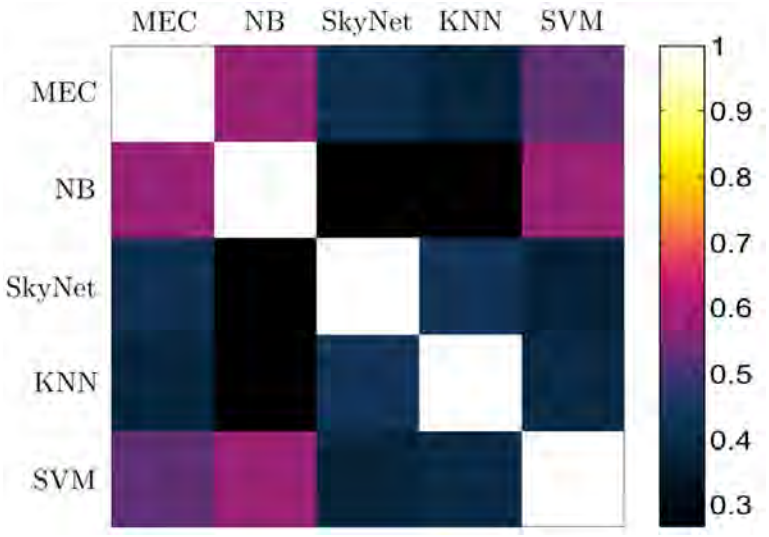


FIGURE 5.2: **Cohen's Kappa.**
The Cohen's Kappa coefficient (κ) value for each pair of classifiers, which measures the overlap in performance between classifiers relative to pure chance (which corresponds to $\kappa \leq 0$). These results can be interpreted with the aid of Table 5.9.

5.5.3 Classifier Performance on Different Visual Classes.

Our classifiers' performance on the respective visual classes (dipoles/saturated, artefacts and real) in the test data set can be seen in Table 5.10. For the recall value calculations here, the true positives (t_p) are associated with objects belonging to a specific visual class that were correctly classified as being either not-real or real, while false negatives (f_n) are associated with incorrect classifications.

For all of our five learning algorithms the performance on artefacts and real objects is better than the performance on dipoles/saturated objects, the latter class therefore being responsible for the lowering of the overall performance of our algorithms. A possible reason behind the poor classification results for this class comes from the observation that dipoles/saturated and real objects can have quite similar features (compare the PCs in Fig. 4.3d with the ones shown in

TABLE 5.10: **Classifier performance on visual classes.**
The recall of the classifiers on the different visual classes of images. The true positives, t_p, for a class correspond to the images in that class that were correctly classified, and f_n corresponds to the images that were incorrectly classified. Recall here has the same meaning as accuracy.

Visual Class	kNN	SkyNet	SVM	MEC	NB
Real	0.90	0.89	0.90	0.92	0.86
Artefact	0.93	0.92	0.90	0.90	0.82
Dip/Sat	0.79	0.76	0.62	0.47	0.38

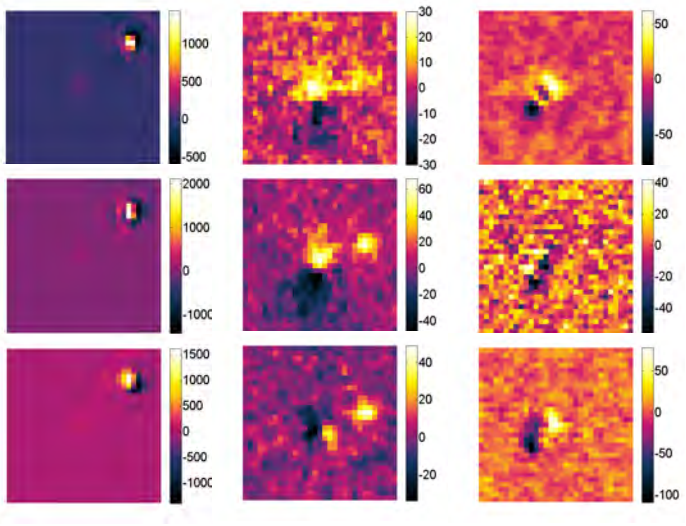
Fig. 4.3b), something that can very likely confuse classifiers. Seeing as many dipoles/saturated objects resemble real objects quite closely, and seeing as real object features are (therefore) very well-represented in the PCs we use for feature extraction, noisy dipoles/saturated objects are very easily misclassified as being real - the noise in their images often result in the loss of their distinctive characteristics, leaving them to appear very similar to noisy real objects.

5.5.4 Incorrectly Classified Images

To form an idea of possible reasons behind object misclassifications, we investigated objects that all five of our learning algorithms classified incorrectly. Fig. 5.3 displays three real objects classified as not-real by all classifiers while Fig. 5.4 displays three not-real objects classified as real by all classifiers.

Fig. 5.3a shows an object having a dim point-like residual in the centre of each of its three images - these dim residuals are the reasons for the "real" label of the object, as they are an indication of the object possibly being a SN. This object's misclassification probably has to do with the fact that all of its images contain a bright dipole-like structure in the top right corner - as we did not restrict the classifiers to only focus on the central pixels of the images, this would be a confusing factor to them. Both of the real objects shown in Fig. 5.3b and 5.3c have dipole-like residuals in all three of their passbands, leaving no mystery as to why all five classifiers mistook them for not-real objects. Originally, the "real" label of the two objects might even have been a mistake made by the human hand scanners, emphasising how erroneous labels can eventually influence the test results.

Fig. 5.4a shows a similar problem, where hand scanners assigned an object to the artefact class that the learning algorithms understandably classified as a real object seeing as there are point-like structures present in all three of its bands. Fig. 5.4b shows an artefact with all three of its images containing masked areas, resulting in the loss of its characteristic spike-like residuals. The remaining point-like residual in the central area of its *i*-band image is probably the reason behind it being misclassified as real by all five classifiers. Fig. 5.4c shows a dipole object, as is



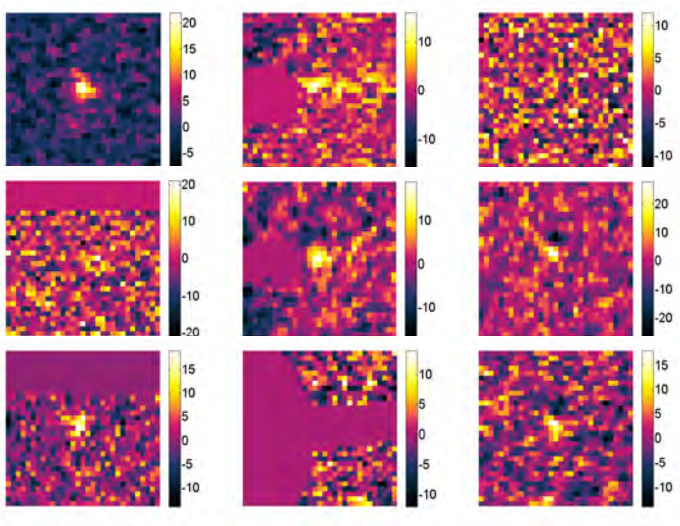
(A) (B) (C)

FIGURE 5.3: **Real objects classified as not-real.**
Examples of objects classified as real by the human hand scanners but classified as not-real by all five machine classifiers. For each object the *g*-band (top), *i*-band (middle) and *r*-band (bottom) image is shown: (a) the dipole-like residuals in the top corner outshining the faint point-like structures probably resulted in misclassification; (b&c) the dipole-like structures observed in the centre of these real object images probably resulted in their misclassification.

made evident to us by the dipole-like structures present in its i and r-band images. The noisy nature of the object's images is possibly what caused all five classifiers to misclassify it as a real object. Furthermore, as was discussed in section 5.5.3, it is often the case that objects from the dipoles/saturated and real class share quite similar features - a possible contributing factor to the object incorrectly being classified as real.

5.5.5 Comparison with Human Classifiers.

As was mentioned in section 3.4.2, fake SNe were injected into the SDSS-II SN survey's pipeline in order to test hand scanner efficiency. The recall performance on the fake SNe, averaged over all the hand scanners, was 0.956 ± 0.010 [176]. For the purpose of comparison, the classification results of our five learning algorithms on the mock SNe in our test data set can be found in Table 5.11. It is clear that three of our classifiers (KNN, SkyNet and MEC) yielded results on the fake SNe that were as good as, or even better than, the average hand scanner recall. Upon inspection of Fig. 5.1, it can furthermore be seen that varying the discrimination threshold and suffering only minor penalties (larger FPR) can result in a recall (TPR) of 96%, even in the case where all test data are used.



(A) (B) (C)

FIGURE 5.4: **Not-real objects classified as real.**
Examples of objects classified as not-real by the human hand scanners that were each classified as being real by all five machine classifiers. For each object the *g*-band (top), *i*-band (middle) and *r*-band (bottom) image is shown: (a&b) these artefacts were probably misclassified due to the fact that they appear more point-like than spike-like; (c) the noisy images of this dipole probably resulted in its misclassification.

TABLE 5.11: **Performance comparison with humans.**
A summary of the performance results of our five classifiers on the fake SNe found in the test set. The results that are equal or better than that obtained by the human hand scanners are in red.

Learning Algorithm	Recall
kNN	**0.96**
SkyNet	**0.96**
SVM	0.94
MEC	**0.96**
NB	0.90

5.5.6 Performance on Spectroscopically Confirmed SNe

Our classifiers' performance on the set of spectroscopically confirmed SNe found in our test data set can be seen in Table 5.12 - our test set contained 135 SNe that were spectroscopically confirmed, of which 110 were SNe Ia and the remaining 19 were comprised of other SNe types (type Ib, Ic and II).

It can be seen that MEC, SVM, SkyNet and kNN yielded good results, with a minimum recall value of 0.90 amongst them - an echo of earlier successes. The weakest learning algorithm of the group continues to be the NB classifier, with a maximum recall value of 0.76 for the case of spectroscopically confirmed Type Ia SNe.

TABLE 5.12: **Classifier performance on spectroscopically confirmed SNe.**
The recall of the various classifiers on the 135 spectroscopically confirmed SNe in the test set (110 SNe Ia and 19 other SNe). The first row shows the performance on the group of SNe Ia, while the second row details the results for the other 19 SNe (excluding SNe Ia). The last row shows the classification results for all 135 spectroscopically confirmed SNe together.

	kNN	SkyNet	SVM	MEC	NB
SN Ia	0.94	0.92	0.90	0.91	0.83
SNe	1.00	1.00	0.89	1.00	0.63
All	0.95	0.93	0.90	0.93	0.80

5.5.7 Classifier Run Time Considerations

The computational run time for training and testing each of our five classifiers with their respective best-performing feature sets, along with the various machines used for their implementation, can be seen in Table 5.13.

It can be seen that, with the exception of SkyNet, all run times were on the order of mere seconds or minutes. SkyNet's long training time ($\sim$ 18 h) might appear to be disadvantageous, but when keeping in mind that classifier training is carried out only once and that thereafter the real-time classification of new objects can take place indefinitely, this becomes an insignificant once-off payment in time. It is therefore the testing time of the classifiers that is of lasting importance - in our case all classifiers classified the 6874 objects in the test set in seconds or minutes. It should be noted that this same task would take a human hand scanner days to complete.

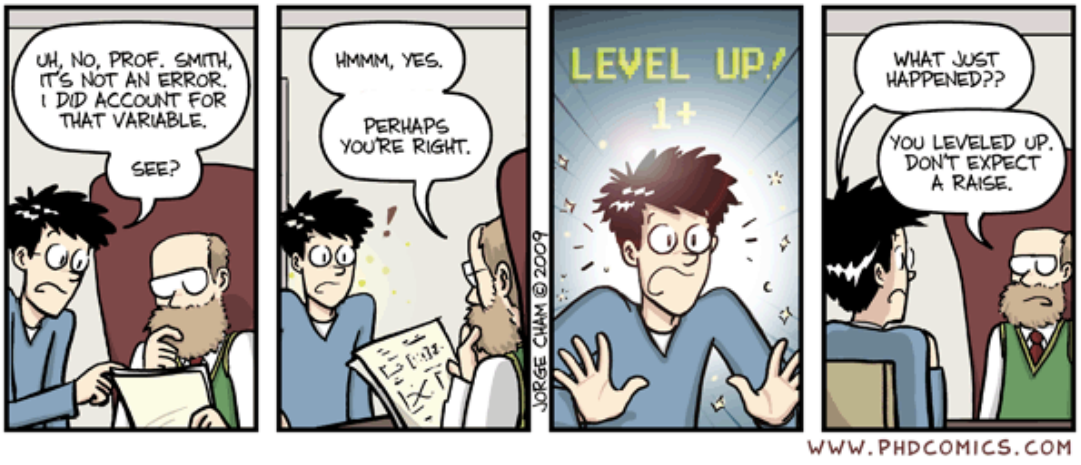
TABLE 5.13: **Classifier run times.**
The training and testing run times for our various classifiers on their respective best-performing feature sets, along with the machines on which they were implemented. Note that MEC does not require any training (see section 5.4.1.2). The desktop computer had a clock speed of 2.4 GHz, the laptop computer had an Intel Core i7 2.4 GHz processor and the cluster was an Intel 2.4 GHz, of which only one core was used (see section 5.4.5).

	Training time	Testing time	Machine
kNN	$\sim$ 20 s	$\sim$ 35 s	Desktop computer
SkyNet	$\sim$ 18 h	$\sim$ 1 s	Computer cluster
SVM	$\sim$ 1.5 min	$\sim$ 10 s	Desktop computer
MEC	N/A	$\sim$ 3 s	Laptop computer
NB	$\sim$ 6 s	$\sim$ 1.5 min	Laptop computer

Even in the unlikely case where machine classifiers and humans take the same amount of time to classify new objects accurately, it would still be advantageous to have the job done by a machine - not only does it enable the human scanner to do other important tasks that can not be automated, but it also ensures that any classification results are reproducible and that human classification biases are avoided. Apart from this, hand scanning will be completely infeasible for surveys like the LSST due to the fact that millions of images will have to be scanned per night.

Chapter 6

Conclusions and Future Work



– "Piled Higher and Deeper" by Jorge Cham
www.phdcomics.com

The search to find answers to the deepest questions we have about the Universe has fueled the collection of data for ever larger volumes of our cosmos. The field of supernova cosmology, for example, is seeing continuous development due to its importance in constraining cosmological parameters, with surveys such as the LSST, DES, Pan-STARRS and the PTF set to produce a vast amount of data that will require new statistical inference and machine learning techniques for processing and analysis.

This vast amount of data creates problems in the analysis pipeline of such SN surveys - when we consider difference images, for example, they should ideally consist of pure noise unless a transient exists in the reference image. In real life, however, instrumental effects cause the occurrence of artefacts, and it is therefore necessary to be able to distinguish between artefacts and objects that might potentially be of interest. This classification into artefacts and real objects has, historically, been carried out by astronomers scanning the images by hand shortly after they have been taken.

In the case of the SDSS-II SN Survey, this practice lead to hand scanners typically sorting hundreds or thousands of images per night. This method of classification has two very apparent problems. The first is the issue of time - assigning humans to a task such as this is very time-consuming, and would be a completely infeasible approach to classification for the large upcoming surveys mentioned above. The second issue is that human classifiers introduce great difficulties in quantifying the biases originating from the different internal decision trees and algorithms in the brain of each scanner.

To succeed in the objectives of future transient surveys, the successful substitution of human hand scanners with machine learning techniques for the purpose of this artefact-transient classification therefore represents a vital frontier. In this thesis we explore this prospect by testing various leading machine learning algorithms and showing that many of them can match the human hand scanner performance in classifying transient imaging data from the SDSS-II SN survey into real objects and artefacts.

In Chapter 2, we introduce the field of machine learning, an area of research concerned with the creation of computer programs that can improve their performance on a given task through experience. We start by discussing the important concepts used in the discipline and describe the three major learning categories that machine learning applications most commonly belong to: supervised, unsupervised and reinforcement learning. In this thesis we are interested in supervised classification, having as its goal the assignment of an input instance to one of a number of mutually exclusive classes, using training sets consisting of both the features and the class values of the training instances. We next supply the theoretical background for the machine learning algorithms and the feature extraction techniques that we employ in later chapters: we discuss the Naïve Bayes (NB) classifier, the k-Nearest Neighbours (kNN) algorithm, the Support Vector Machine (SVM) algorithm and Artificial Neural Networks (ANNs) as classification algorithms, and Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) as feature extraction techniques.

Chapter 3 introduces SN cosmology - it describes the main categories of SNe and discusses the importance of using Type Ia SNe as standard candles for the constraining of cosmological models. The chapter also discusses the data deluge expected from upcoming SN surveys, addresses the limitations and problems introduced by using human hand scanners for artefact/real classification (as mentioned above) and proposes the substitution of human classifiers with machine learning algorithms. It then proceeds to give an outline of the science goals and technical aspects of the SDSS-II SN Survey, after which it describes the data we draw from it for our classification purposes, the classification systems we employ, the splitting of our data into a training, validation and a withheld test set and our classification goal - to match the performance of the SDSS human hand scanners in distinguishing between real objects (transients) and not-real objects (artefacts, dipoles and saturated stars).

In Chapter 4 we discuss the pre-processing and feature extraction techniques applied to our data in order to produce the various feature sets we employ for classification purposes in Chapter 5. We carry out PCA on each one of our visual classes in the training set individually in order to obtain a set of PCs for each visual class - these PCs can then be used to reconstruct a test object for the purpose of calculating its reconstruction error features. PCA was also carried out on our full training set in order to obtain a set of PCs for the extraction of PC weights from test objects. Finally, we applied LDA to our training data for the purpose of calculating an LDA component for each test object. By varying the number of PC weights, by either including or omitting the LDA component, by either normalising the features in the feature set or leaving them non-normalised, and by extracting features from either 31×31 or 51×51 pixel images, a total of 56 different feature sets were obtained with which to optimise our various classifiers with.

In Chapter 5 we start off by discussing the performance measures we use for judging the classification results of our learning algorithms - accuracy, recall, precision, the F_1-score and the AUC value. We then proceed to discuss the training, validation and testing procedures for our classifiers. In general, all our algorithms are initially trained and validated with all 56 different feature sets. During this phase the most successful feature set for each classifier is determined and, where necessary, classifier parameters are also optimised. The algorithms are then submitted to the final testing phase during which each classifier is tested with the feature set (now taken from the withheld test set) that proved most optimal for it during the training and validation phase. The chapter next discusses the implementation and the training, validation and final testing of each of our five classifiers: a very simple method referred to as Minimum Error Classification (MEC), a kNN and SVM algorithm obtained from the work of Pedregosa et al. [166], an ANN algorithm, SkyNet, obtained from the work of Graff et al. [88] and an NB classifier.

The classification results are then summarised, compared and analysed. After additionally obtaining ROC curves and AUC values for all five classifiers, it is found that kNN, a very basic nonparametric learner, is the best-performing classifier for our problem with regard to accuracy (89%), recall (90%) and the AUC value (94%), our chosen measures for final performance evaluation. SkyNet and SVM follow closely behind. A study of inter-classifier agreement is next conducted, and it is found that none of the best-performing classifiers (kNN, SkyNet and SVM) are highly correlated in their classifications, an indication that combining different classifiers' predictions in an ensemble classifier might yield improved classification results. Classifier performance on the different visual classes is investigated next, and it is found that all five classifiers perform better on artefacts and real objects than on dipoles/saturated objects. The latter class is therefore responsible for the lowering of the overall performance of our algorithms. We also discuss objects that were incorrectly classified by all our classifiers, and speculate as to the possible reasons why that might have happened - original mislabelling by the hand scanners, masking and noise obscuring the distinctive features of objects and the presence of other objects in images confusing the classifiers are considered. When comparing our results with those of the human hand scanners on

the set of fake SNe in our test data, it is seen that three of our classifiers (kNN, SkyNet and MEC) match their recall performance of $\sim 96\%$, meaning that our goal of matching human performance is achieved. We then investigate the algorithms' performance on the spectroscopically confirmed SNe in the test set, and find that all our classifiers (apart from NB) yielded excellent results, with kNN yielding the highest recall value of 95%. Finally, the various training and testing run times of our classifiers are discussed - it is found that, with the exception of SkyNet's training time of ~ 18 h, all other run times were on the order of mere seconds or minutes, much faster than human hand scanners would be able to accurately classify the same data sets.

This thesis opens many possible avenues for future research. By also including the search images of objects in the classification process (and not only making use of difference images), we will have access to the host galaxy information of objects - information that might improve classification performance. It might also prove useful to include information on the history of objects. We are currently able to match human hand scanners by using only difference images, while the hand scanners themselves had access to difference and search images as well as object history - it would be interesting to compare our results once we make use of the extra information they employ. More possibilities for research include the use of multi-epoch data for classification and the combination of algorithms into an ensemble classifier (as mentioned above) to investigate the possibility of any further improvements in the classification performance. With all the above information added to the classification process, it would also be interesting to investigate the possibility of distinguishing between different subclasses of transients.

The use of machine learning algorithms for data processing, classification and analysis in astronomy is going to play a role of ever-increasing importance as the amount of data we work with increase. Our future ability to effectively use large repositories of data may depend largely on how effective our machine learning capabilities are - making this an important, challenging and rewarding problem to solve in the coming years.

Appendix A

Mathematical Background on Support Vector Machines

Introduction

This appendix contains additional information and derivations related to the support vector machine algorithm, discussed in section 2.2.3. The main source is [92].

A.1 Derivation of Support Vector Machine Constraints for Linearly Separable Training Data

Following from section 2.2.3.1, we are looking for solutions resulting in the correct classification of all training instances, and therefore have $t_n y(\mathbf{x}_n) > 0$ for all n. It can be shown that the perpendicular distance between a point $\mathbf{x}_n$ and the decision boundary can be given as

$$r = \frac{y(\mathbf{x})}{||\mathbf{w}||}. \tag{A.1}$$

By using

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}) + b, \tag{A.2}$$

defined in section 2.2.3.1, we can therefore write Eq. A.1 as

$$\frac{t_n y(\mathbf{x}_n)}{||\mathbf{w}||} = \frac{t_n(\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) + b)}{||\mathbf{w}||}. \tag{A.3}$$

Seeing as the margin is defined to be the smallest perpendicular distance between the decision boundary and any instance $\mathbf{x}_n$ in the training data set, and seeing as we want to maximise the margin with respect to $\mathbf{w}$ and b, we have to solve the following:

$$\underset{\mathbf{w},b}{\operatorname{argmax}} \left\{ \frac{1}{||\mathbf{w}||} \min_n \left[t_n \big(\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) + b \big) \right] \right\} \tag{A.4}$$

It should be noted that $\mathbf{w}$ is not dependent on n. Finding a direct solution to Eq. A.4 is very complex, and so we try instead to find an equivalent problem that we can solve more easily. We see that when the two parameters $\mathbf{w}$ and b are rescaled such that $\mathbf{w} \to \beta\mathbf{w}$ and $b \to \beta b$, the distance between the decision boundary and any instance $\mathbf{x}_n$, given by Eq. A.3, remains unchanged. This allows us to set

$$t_n \big(\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) + b \big) = 1 \tag{A.5}$$

for the instance $\mathbf{x}_n$ closest to the decision boundary, resulting in all training instances satisfying the constraints

$$t_n \big(\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) + b \big) \geq 1, \qquad n = 1, ..., N \tag{A.6}$$

referred to as the decision hyperplane's canonical representation. This has been taken into account in Fig. 2.3. For data instances for which the equality in Eq. A.6 holds, the constraints are referred to as being *active*, while for the rest of the instances the constraints are said to be *inactive*. Seeing as there is always an instance closest to the decision hyperplane, there is always one active constraint at least, and two active constraints if the margin is maximised. Taking the above into account, we now only need to maximise $||\mathbf{w}||^{-1}$ in order to solve Eq. A.4. Seeing as this is the same as minimising $||\mathbf{w}||^2$, we are simply left with having to solve

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \frac{1}{2}||\mathbf{w}||^2, \tag{A.7}$$

while taking into account the constraints set out in Eq. A.6. This is now a *quadratic programming* problem[1]. The inclusion of the 1/2 factor in Eq. A.7 will be made clear later. It might appear that b is not part of the optimisation problem any longer, but it is in fact now just implicitly determined by the constraints, requiring that when changes are made to $||\mathbf{w}||$ it is compensated by changing b.

A.2 Determination of Support Vector Machine Parameters for Linearly Separable Training Data

Following from section 2.2.3.1, because we are faced with having to solve a constrained optimisation problem, we use the Lagrange multipliers[2] $a_n \geq 0$ (one for each constraint in Eq. A.6), resulting in the Lagrangian function

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{n=1}^{N} a_n \left\{ t_n \left(\mathbf{w}^{\mathrm{T}} \phi(\mathbf{x}_n) + b \right) - 1 \right\} \tag{A.8}$$

Here, $\mathbf{a} = (a_1, ..., a_N)^{\mathrm{T}}$ - it should also be noted that we are here minimising with respect to b and $\mathbf{w}$ while maximising with respect to $\mathbf{a}$. By taking the partial derivative of Eq. A.8 with respect to b and $\mathbf{w}$, and setting those relations equal to zero, these two conditions are obtained:

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{w}_n) \tag{A.9}$$

$$0 = \sum_{n=1}^{N} a_n t_n \tag{A.10}$$

By replacing $\mathbf{w}$ and b in Eq. A.8 with the above two relations, we find the maximum margin problem's *dual representation*, shown in Eq. A.11. This then has to be maximised with respect to $\mathbf{a}$:

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{A.11}$$

This maximisation will be carried out subject to a set of constraints, shown in Eq. A.12 and A.13:

[1] A quadratic programming problem is one in which we try to optimise a quadratic objective function while taking certain linear inequality constraints into consideration.

[2] For information on Lagrange multipliers, see Appendix E in Bishop's book [92].

$$a_n \geq 0, \qquad n = 1, ..., N \tag{A.12}$$

$$\sum_{n=1}^{N} a_n t_n = 0. \tag{A.13}$$

In Eq. A.11, the kernel function is defined by

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}} \phi(\mathbf{x}') \tag{A.14}$$

as noted earlier in section 2.2.3.2. This optimisation problem is again an example of a quadratic programming problem.

If a quadratic programming problem has M variables, it generally requires a solution with computational complexity $O(M^3)$. In transforming the original optimisation problem in Eq. A.7 into the dual problem in Eq. A.11, we have moved from minimising over M variables to maximising over N variables, where M is the number of fixed basis functions and N is the number of training data instances. It may appear that the change to the dual problem is disadvantageous when looking at situations where M is smaller than N. However, it allows a reformulation that makes use of kernel functions, enabling the classifier to be applied successfully to feature spaces having dimensionality larger than the number of training instances N. This holds even for feature spaces having infinite dimensionality. The formulation given above further also clarifies the purpose of the constraint specifying the positive definiteness of the kernel function $k(\mathbf{x}, \mathbf{x}')$ (see section 2.2.3.2), as it causes the Lagrangian function $\widetilde{L}(\mathbf{a})$ to be bounded below.

The sign of $y(\mathbf{x})$ (given by Eq. A.2) is evaluated for the classification of new input test instances when the training phase is completed and the model determined. To express this in terms of the kernel function and the parameters $\{a_n\}$, we use Eq. A.9 to substitute for $\mathbf{w}$ in Eq. A.2 in order to obtain

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b. \tag{A.15}$$

It can be shown that a constrained optimisation of the form shown in Eq. A.15 satisfies the *Karush-Kuhn-Tucker* (KKT) [177, 178] conditions (see section A.5). In the case of our problem, this means that the optimisation of Eq. A.15 requires the following constraints to hold:

$$a_n \geq 0 \tag{A.16}$$

$$t_n y(\mathbf{x}_n) - 1 \geq 0 \tag{A.17}$$

$$a_n\{t_n y(\mathbf{x}_n) - 1\} = 0 \tag{A.18}$$

From this it can be seen that for every $\mathbf{x}_n$, either $a_n = 0$ or $t_n y(\mathbf{x}_n) = 1$. In the latter case, instances are support vectors (defined in section 2.2.3.1). Having established a value for $\mathbf{a}$ by solving the problem in Eq. A.11, we now want to determine b's value. By keeping in mind that the relation $t_n y(\mathbf{x}_n) = 1$ holds for any support vector, substituting for $y(\mathbf{x})$ using Eq. A.15 we have that

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right) = 1, \tag{A.19}$$

where $\mathcal{S}$ is the set of support vector indices. Instead of determining b through the use of a random support vector $\mathbf{x}_n$, we rather focus on a solution that is more stable numerically. This is done by multiplying both sides of Eq. A.19 with t_n (noting that $t_n^2 = 1$) and then taking the average of the equations over the entire set of support vectors. Finding b then results in Eq. A.20, where the support vectors' total number is denoted by $N_{\mathcal{S}}$.

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right). \tag{A.20}$$

A.3 Determination of Support Vector Machine Parameters for Linearly Non-Separable Training Data

Following from section 3.2, we need to solve for the following Lagrangian function

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}||\mathbf{w}||^2 + C \sum_{n=1}^{N} \zeta_n - \sum_{n=1}^{N} a_n\{t_n y(\mathbf{x}_n) - 1 + \zeta_n\} - \sum_{n=1}^{N} \mu_n \zeta_n, \tag{A.21}$$

where $\{a_n \geq 0\}$ and $\{\mu_n \geq 0\}$ are Lagrange multipliers, subject to the KKT conditions

$$a_n \geq 0 \tag{A.22}$$
$$t_n y(\mathbf{x}_n) - 1 + \zeta_n \geq 0 \tag{A.23}$$
$$a_n(t_n y(\mathbf{x}_n) - 1 + \zeta_n) = 0 \tag{A.24}$$
$$\mu_n \geq 0 \tag{A.25}$$
$$\zeta_n \geq 0 \tag{A.26}$$
$$\mu_n \zeta_n = 0 \tag{A.27}$$

with $n = 1, \ldots, N$. Substituting $y(\mathbf{x})$ in Eq. A.21 for its definition, given in Eq. A.2, and setting the partial derivatives of the Lagrangian with respect to $\mathbf{w}$, b and $\{\xi_n\}$ equal to zero, the following is obtained:

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{n=1}^{N} a_n t_n \phi(\mathbf{x}_n) \tag{A.28}$$
$$\frac{\partial L}{\partial b} = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} a_n t_n = 0 \tag{A.29}$$
$$\frac{\partial L}{\partial \zeta_n} = 0 \quad \Rightarrow \quad a_n = C - \mu_n \tag{A.30}$$

Eq. A.28, A.29 and A.30 is then used for the purpose of eliminating $\mathbf{w}$, b and $\{\xi_n\}$ in Eq. A.21 to obtain its dual representation, shown in Eq. A.31.

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \tag{A.31}$$

Although this looks similar to Eq. A.11, the difference lies in the constraints it is subject to. Firstly, noting Eq. A.30 and the Lagrange multiplier $\mu_n \geq 0$, it can be seen that $a_n \leq C$. Further taking into account the fact that a_n is also a Lagrangian multiplier and therefore constrained to $a_n \geq 0$, we find that Eq. A.31 should be minimised with respect to $\{a_n\}$ while taking the constraints

$$0 \leq a_n \leq C \tag{A.32}$$
$$\sum_{n=1}^{N} a_n t_n = 0 \tag{A.33}$$

into account, where $n = 1, \ldots, N$. Eq. A.32 are called *box constraints*. As before, this is a quadratic programming problem. By substituting Eq. A.28 into Eq. A.2, it is found that, as

before, classifications for new test data are made through the use of Eq. A.15, given again here for convenience.

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b \tag{A.34}$$

As before, we note that training instances having $a_n = 0$ will play no role in the above model, with the rest of the instances being the support vectors with $a_n > 0$. Using Eq. A.24, the support vectors must therefore satisfy

$$t_n y(\mathbf{x}_n) = 1 - \zeta_n \tag{A.35}$$

When $a_n < C$, it can be seen from Eq. A.30 that $\mu_n > 0$. This, together with Eq. A.27, means that the slack variables must therefore be of value $\xi_n = 0$, meaning that instances with such a value of a_n will fall on the margin itself. Instances having $a_n = C$ can fall inside the margin and can either be correctly classified or misclassified based on whether the value of the slack variable is $\xi_n \leq 1$ or $\xi_n > 1$, respectively.

Lastly, to find the bias b in Eq. A.2, we remark that for support vectors having $0 < a_n < C$, the slack variables are $\xi_n = 0$. Substituting this into Eq. A.35 gives

$$t_n y(\mathbf{x}) = 1 \tag{A.36}$$

and subsequently substituting for $y(\mathbf{x})$ using Eq. A.15 and noting that only support vectors have any influence on the predictive model, we can write

$$t_n \left(\sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b \right) = 1 \tag{A.37}$$

By averaging to get a stable solution, as before, we find that

$$b = \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} \left(t_n - \sum_{m \in \mathcal{S}} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right) \tag{A.38}$$

Here, the set of indices of the instances having $0 < a_n < C$ is denoted by $\mathcal{M}$.

A.4 Constructing New Kernels

One way in which new valid kernels can be constructed, is to use simpler kernels as building blocks to build them. Given the kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the properties listed in Eq. A.39–A.48 can be used to construct new kernels.

$$k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}') \tag{A.39}$$
$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \tag{A.40}$$
$$k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}')) \tag{A.41}$$
$$k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}')) \tag{A.42}$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \tag{A.43}$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \tag{A.44}$$
$$k(\mathbf{x}, \mathbf{x}') = k_3(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')) \tag{A.45}$$
$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' \tag{A.46}$$
$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \tag{A.47}$$
$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a)k_b(\mathbf{x}_b, \mathbf{x}'_b) \tag{A.48}$$

Here, $\mathbf{x}$ is an input vector, $c > 0$ is a constant, $f(\cdot)$ denotes any function, $q(\cdot)$ denotes a polynomial having nonnegative coefficients and $\boldsymbol{\phi}(\mathbf{x})$ is a transformation from $\mathbf{x}$ to $\Re^M$. The function $k_3(\cdot, \cdot)$ is a valid kernel in $\Re^M$, $\mathbf{A}$ is a positive semidefinite symmetric matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernels, each over their own respective spaces.

A.5 Karush-Kuhn-Tucker Conditions

Karush-Kuhn-Tucker (KKT) conditions arise when a function $f(\mathbf{x})$ has to be maximised while taking a constraint $g(\mathbf{x}) \geq 0$ into account. A solution is found by optimising the Lagrangian function

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}) \tag{A.49}$$

where λ is a Lagrange multiplier, with respect to λ and $\mathbf{x}$, subjected to the following constraints or conditions:

$$g(\mathbf{x}) \geq 0 \tag{A.50}$$

$$\lambda \geq 0 \tag{A.51}$$

$$\lambda g(\mathbf{x}) = 0 \tag{A.52}$$

Appendix B

Artificial Neural Network Training

Introduction

This appendix contains additional information and derivations related to the training of artificial neural networks, discussed in section 2.2.4.4. The main source was [92].

B.1 Network Training

A simple method for determining the parameters $\mathbf{w}$ of the simple feed-forward neural network is by minimising the error function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} ||\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n||^2 \tag{B.1}$$

where $\{\mathbf{x}_n\}$, with $n = 1, \ldots, N$ is the set of input training feature vectors and where the set $\{\mathbf{t}_n\}$ consists of the corresponding target values. A more general outlook on network training, however, can be provided by first interpreting network outputs in a probabilistic manner, in the process motivating the choice of activation function used for the output layer as well as the choice of the specific error function to minimise.

The problem we consider here is one in which an input instance must be classified as being one of K distinct classes. The target values $t_k \in \{0, 1\}$ are encoded with a 1-of-K scheme in order to indicate the class - to illustrate this, see Eq. B.2 below for the case where $K = 3$. Three objects (o_1, o_2 and o_3) are shown: they belong to classes 1 (left), 2 (middle) and 3 (right).

$$o_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad o_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad o_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{B.2}$$

Network outputs are probabilistically interpreted as being

$$y_k(\mathbf{x}, \mathbf{w}) = P(t_k = 1|\mathbf{x}). \tag{B.3}$$

This is done through the application of the *softmax* function[1] to the output activations a_k, as is shown here:

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))} \tag{B.5}$$

with $j = 1, \ldots, K$, where $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$. The error function, given by the negative of the log likelihood, is the cross-entropy of the output and target values, and is as follows:

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}) \tag{B.6}$$

This is then the error function that has to be minimised in order to determine $\mathbf{w}$. $E(\mathbf{w})$, being a smooth and continuous function, will have its minimum value at some point in the weight space where its gradient is equal to zero, such that

$$\nabla E(\mathbf{w}) = 0 \tag{B.7}$$

and the goal here is to determine a vector $\mathbf{w}$ in order for the above to be true. Because $E(\mathbf{w})$ has a nonlinear dependence on the $\mathbf{w}$ parameters, many solutions in weight space will correspond with the gradient vanishing - finding the global minimum may not be needed for the successful

[1] In probability theory, the softmax function is applied to a K-dimensional vector $\mathbf{s} = (s_1, \ldots, s_K)$ of real values to transform it into the vector $\mathbf{r}$ of the same dimension, where each of $\mathbf{r}$'s real-valued elements now falls in the range $(0, 1)$. The softmax function is as follows:

$$r_k = \frac{\exp(s_k)}{\sum_j \exp(s_j)}, \quad \text{with } k = 1, \ldots, K \text{ and } j = 1, \ldots, K \tag{B.4}$$

Since the elements of the new vector $\mathbf{r}$ have values between 0 and 1 and sum to 1, they now form a categorical probability distribution.

implementation of the neural network, but comparing a number of local minima might be required for the finding of an adequately accurate solution.

We employ iterative numerical methods to find a solution to Eq. B.7. Mostly, methods for this task require that some initial vector $\mathbf{w}^{(0)}$ be chosen, and the weight space is then traversed in iterative steps

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \tag{B.8}$$

in order to reach a minimum in the error function, where the gradient will be zero. Here, τ denotes the iteration step. We next look at the Taylor expansion of the error function around a point in weight space $\widehat{\mathbf{w}}$, as this local quadratic approximation of $E(\mathbf{w})$ gives insight into the problem and the methods used for solving it. The approximation is shown in Eq. B.9

$$E(\mathbf{w}) \simeq E(\widehat{\mathbf{w}}) + (\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{b} + \frac{1}{2}(\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{H}(\mathbf{w} - \widehat{\mathbf{w}}) \tag{B.9}$$

where $\mathbf{b}$ denotes the error function's gradient at $\widehat{\mathbf{w}}$, as

$$\mathbf{b} \equiv \nabla E|_{\mathbf{w}=\widehat{\mathbf{w}}} \tag{B.10}$$

and where $\mathbf{H}$ is called the Hessian matrix ($\mathbf{H} = \nabla\nabla E$) with elements

$$(\mathbf{H})_{ij} \equiv \frac{\partial E}{\partial w_i \partial w_j}|_{\mathbf{w}=\widehat{\mathbf{w}}} \tag{B.11}$$

The local approximation to E's gradient, from Eq. B.9, is

$$\nabla E \simeq \mathbf{b} + \mathbf{H}(\mathbf{w} - \widehat{\mathbf{w}}) \tag{B.12}$$

We use this gradient information to define the weight step update in Eq. B.8 as a step towards the negative gradient of the error function, so that we continuously move closer to a vector $\mathbf{w}$ where the gradient is zero and we therefore have a minimum of the function. The step update is as follows:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E(\mathbf{w}^{(\tau)}) \tag{B.13}$$

where $\eta > 0$ is called the *learning rate*, and is used to adjust the size of the steps in weight space. Because each step moves the weight vector in the direction that the rate at which the error function decreases is largest, this method is called *gradient descent*. With each new step ∇E has to be re-evaluated and the whole training set processed to do so. Methods such as this, where the whole training set is used at the same time in order to determine a single step in weight space, are referred to as *batch* methods. Although intuitively pleasing, gradient descent is a rather poor algorithm, and there are faster and more robust batch optimisation methods - *quasi-Newton* methods and *conjugate gradients* being some of them [179–181].

For gradient descent batch methods, one often has to run the algorithm a few times in order to get a solution yielding a sufficient minimum, each run starting with a different random weight vector $\mathbf{w}^{(0)}$. This can be a very time-consuming process. Le Cun et al. [182], however, introduced a *Single-step* (or *on-line*) gradient descent version that proved more practical for the purposes of training networks with the use of large training sets. Single-step methods, in contrast to batch methods, make adjustments to the weight vector directly after being presented with a single training instance. Error functions for these methods, founded on maximum likelihood for independent training instances, consist of the summation of the errors for each of the training data instances (where N is the total number of instances), and is expressed as

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \tag{B.14}$$

The update rule for on-line gradient descent is then

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}) \tag{B.15}$$

Advantages of single-step gradient descent, as compared to the batch version, is that it can possibly escape local minima and is able to deal with redundancy in the training data set in a more efficient way.

B.2 Error Backpropagation

This section discusses an efficient method for evaluating ∇E called *backpropagation*, an approach that involves error-related information being sent forwards and backwards in a neural network. Many learning methods follow an iterative scheme for error function minimisation, with the weight vector being updated after each iterative step, as seen in the previous section. There are two stages involved in each step being taken. During the first stage the error function's derivatives

must be evaluated with respect to the weight parameters. As backpropagation provides an efficient way of evaluating these derivatives, and seeing that it is during this stage that the backwards propagation of errors takes place, the term "backpropagation" will be used specifically with regard to the evaluation of error function derivatives. The second stage involves the use of the computed derivatives to update the weight vector - the simplest way to do this involves the method of gradient descent and was introduced by Rumelhart et al. [183].

For the derivation of the backpropagation algorithm, we consider a general setup: an arbitrary feed-forward neural network topology, activation functions that are nonlinear and differentiable and a wide range of error functions. Most practical error functions have the form shown in Eq. B.14, given again here as

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \tag{B.16}$$

We now tackle the problem of determining $\nabla E_n(\mathbf{w})$. We start with a simple network model where outputs, denoted by y_k, form linear combinations of inputs, denoted by x_i, as

$$y_k = \sum_i w_{ki} x_i \tag{B.17}$$

and where the error function, for the nth training input instance, looks as follows:

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \tag{B.18}$$

Here, $y_{nk} = y_k(\mathbf{x}_n, \mathbf{w})$. Taking the gradient of E_n with respect to the weight w_{ji}, we have

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \tag{B.19}$$

which constitutes the product of an error term $(y_{nj} - t_{nj})$ corresponding to the output of a connection w_{ji} and the input of the connection, x_{ni}. We now extend these results to more intricate multilayer feed-forward neural networks where, in the general case, every node determines a linear combination of its input as

$$a_j = \sum_i w_{ji} z_i \tag{B.20}$$

with z_i being either an input or a previous node's output sending a connection from node i to node j, and where w_{ji} is that connection's corresponding weight. To determine the output z_j associated with node j, a_j is transformed using an activation function $g(\cdot)$, as follows:

$$z_j = g(a_j) \tag{B.21}$$

For each training instance, we will suppose that the output of every node in the network has been calculated by using Eq. B.20 and B.21 successively - this is referred to as *forward propagation*, as information flows from the network's input nodes to its output nodes.

The next step is to evaluate E_n's derivative with respect to the weight w_{ji}. Although the outputs of the network's nodes depend on a specific input instance, we drop the subscript n to simplify the notation. E_n depends on w_{ji} only via a_j, and the partial derivative chain rule can therefore be used to give

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \tag{B.22}$$

Next, the following notation is introduced:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \tag{B.23}$$

Here, δ is called an *error*. From Eq. B.20 we see that

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \tag{B.24}$$

and by substituting Eq. B.24 and B.23 into Eq. B.22, we arrive at

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \tag{B.25}$$

From Eq. B.25 it is seen that the derivative of E_n can be calculated merely by multiplying the δ-value for the node at the output side of the link associated with the weight w_{ji} with the z_i-value for the node at the input side of the link. This formula then takes a form similar to that determined for the simpler case, given in Eq. B.19. All that is necessary to calculate the derivatives, therefore, is to determine δ_j for all hidden and output nodes in the neural network, and to then use Eq. B.25. For the network's output nodes, we have

$$\delta = y_k - t_k \tag{B.26}$$

Using the partial derivative chain rule once again, we find δ_j for the hidden nodes to be

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{B.27}$$

where the sum is over all nodes k, where k labels either hidden or output nodes, receiving a connection from node j. Fig. B.1 shows the configuration of weights, nodes and parameters. Eq. B.27 uses the fact that changes in a_j causes changes in E_n through changes in a_k. Substituting Eq. B.23 into Eq. B.27 and using Eq. B.20 and B.21, we arrive at the backpropagation equation given in Eq. B.28, which shows us that in order to determine δ for a certain hidden node, the δ's of nodes higher in the network should be propagated backwards.

$$\delta_j = g'(a_j) \sum_k w_{kj} \delta_k \tag{B.28}$$

In the case of batch methods, evaluating the total error's derivative can be done by carrying out the steps discussed above for each training instance and then summing, as follows:

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}} \tag{B.29}$$

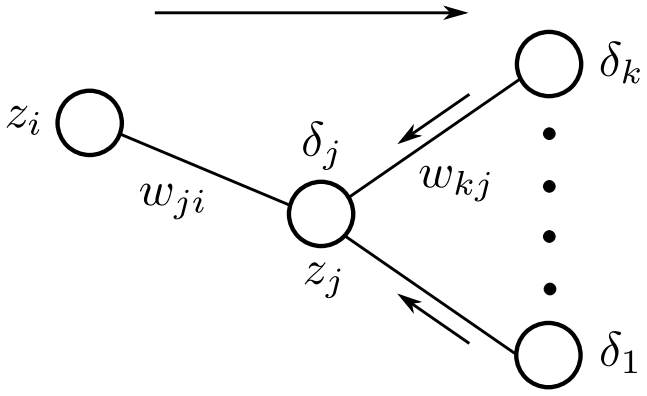


FIGURE B.1: **Backpropagation.**
This illustration helps with the interpretation of Eq. B.28 - where δ_j of a hidden node j is calculated by backpropagating the δ-values from the nodes k receiving connections from j. The top arrow depicts the flow of information during forward propagation, while the two smaller arrows depicts the flow of error information during backpropagation.

Appendix C

Hand-Scanner Classification

This appendix assists in describing the classification system used by the SDSS hand-scanners during visual inspection of images (see section 3.4.1). While Table 3.2 describes each of the different classes and Fig. 3.5 shows a simplified decision tree depicting the classification process followed by the hand-scanners, Table C.1 and C.2 show how the objects in each of these classes look like. Table C.1 shows six r-band difference images from each of the ten classes and also illustrate the difference in image-quality based on SNR, while Table C.2 shows difference images of objects from each class in all three bands (g, r and i).

TABLE C.1: **SDSS hand-scanner classes for different SNR values.**
The ten classes, described in Table 3.2, used by the SDSS hand-scanners during visual classification. For each class, six r-band images of objects are shown - three of which have an SNR above 40, and three having an SNR below 20 (representing roughly 80 % of the dataset).

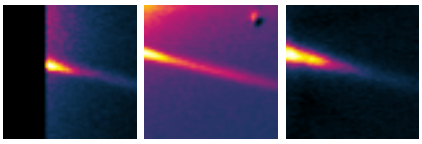
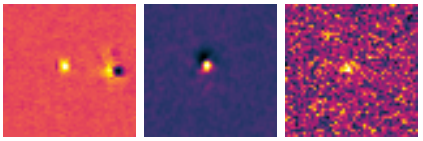
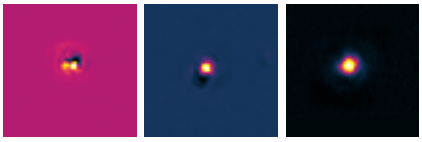
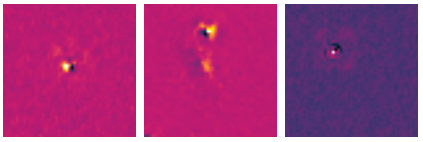
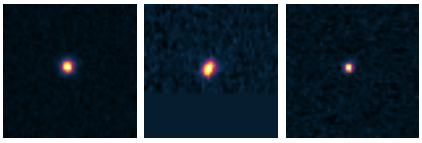
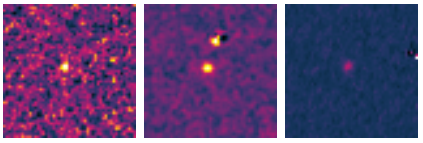
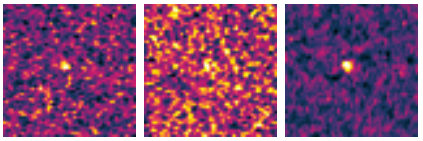
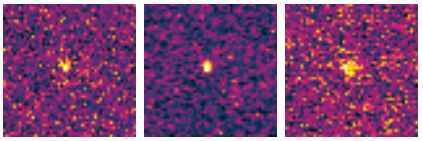
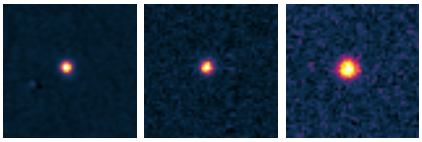
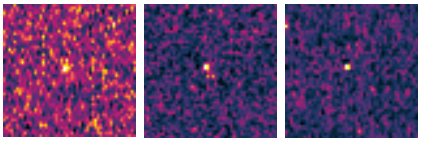
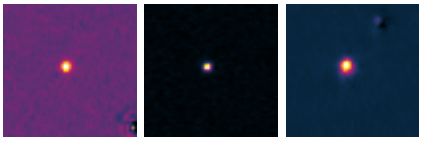
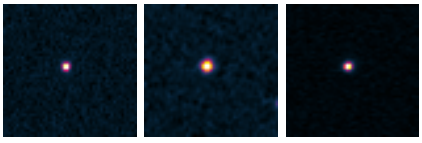
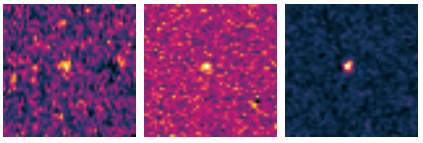
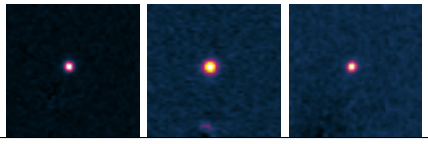
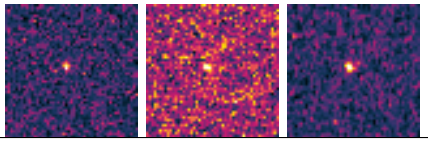
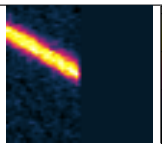
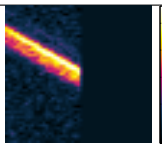
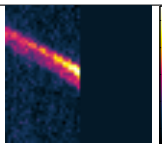
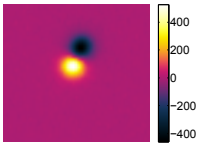
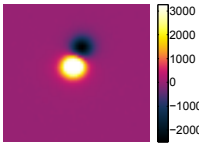
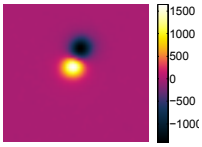
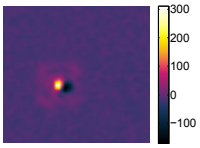
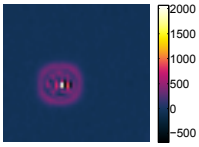
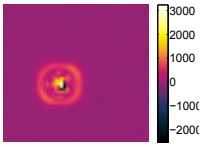
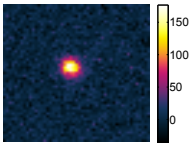
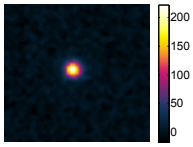
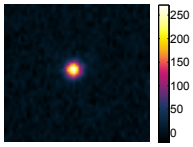
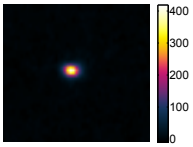
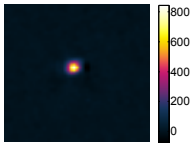
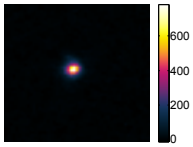
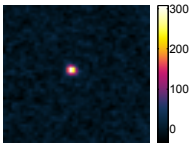
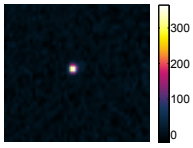
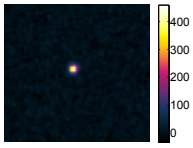
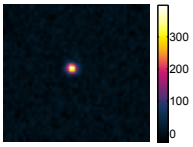
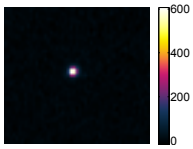
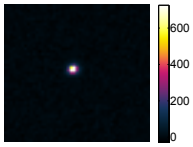
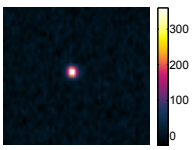
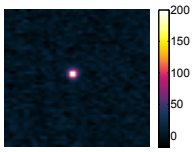
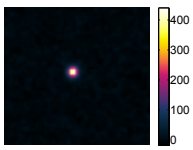
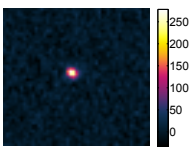
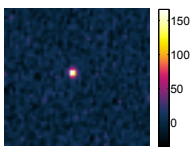
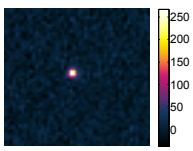
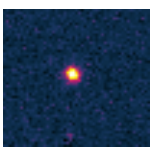
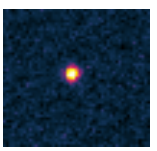
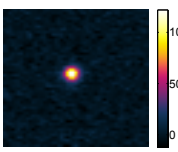
Original Class	Good Quality (SNR > 40)	Bad Quality (SNR < 20)
Artefact		
Dipole		
Saturated Star		
Moving		
Variable		
Transient		
SN Other		
SN Bronze		
SN Silver		
SN Gold		

TABLE C.2: **SDSS hand-scanner classes in different bands.**
The ten classes, described in Table 3.2, used by the SDSS hand-scanners during visual classification. For each class, one object is shown in all three bands (g, r and i).

Original Class	g-band	i-band	r-band
Artefact			
Dipole			
Saturated Star			
Moving			
Variable			
Transient			
SN Other			
SN Bronze			
SN Silver			
SN Gold			

Appendix D

Implementation of Naïve Bayes

This appendix shows the MATLAB® implementation of our Naïve Bayes (NB) algorithm. It shows six functions: Main.m (section D.1), the main program from which all the functions necessary for the training and testing of NB are called; BAYES_INPUT.m (section D.2), generating the required inputs for the training of NB; BINS.m (section D.3), responsible for the binning calculations of the algorithm; TRAIN_BAYES.m (section D.4), the function used to train NB; TEST_BAYES.m (section D.5), the function used for testing and RESULTS.m (section D.6), responsible for calculating the performance metrics. All of these functions are further described in their respective comments sections.

D.1 The Main Program

```
1  %Author: Lise du Buisson
2  %This is the main program from which NB will be run.
3  %
4  %The original input .txt files whose filenames are specified below (look at
5  %input_train, input_test and input_other) details one object instance per row
6  %and have the following comma-separated columns:
7  %   ID | f1 | f2 | ... | fn | c1 | c2 | c3 |
8  % - ID is the object's identification number
9  % - f1, ..., fn are the different feature values
10 % - c1 is a class system used by a collaborator - the details are unimportant here
11 % - c2 is the true class values ("1", "2" or "3") to distinguish between our
12 %   three visual classes
13 % - c3 is true main class values ("0" or "1") to distinguish between not-real
14 %   and real objects (this will be used for training and testing)
15 %-------------------------------------------------------------------------------
16
```

```
%counters
numberPCA = [50];
LDAf = 'False';
LDAt = 'True';
normf = 'False';
normt = 'True';
counter = 1;

%read, train and test all the different feature sets
for pca = 1:length(numberPCA)
    for lda = 1:1
        if lda == 1
            LDA = LDAf;
        else
            LDA = LDAt;
        end
        for nrm = 1:1
            if nrm == 1
                NORM = normf;
            else
                NORM = normt;
            end

            %display counter:
            SETNAME = strcat(num2str(counter), '___PCA',
                num2str(numberPCA(pca)),'_lda', LDA, '_norm', NORM);
            disp(SETNAME);

            %construct filenames:
            %input training, validation and testing set:
            input_train = strcat('features75percent\TRAINING_imageSize31_PCA',
                num2str(numberPCA(pca)), '_LDA', LDA, '_FeatureNormalize',
                NORM, '_features.txt');
            input_test = strcat('features25percent\TEST_imageSize31_PCA',
                num2str(numberPCA(pca)), '_LDA', LDA, '_FeatureNormalize',
                NORM, '_features.txt');
            input_other = strcat('features75percent\TEST_imageSize31_PCA',
                num2str(numberPCA(pca)), '_LDA', LDA, '_FeatureNormalize',
                NORM, '_features.txt');
            %classified input training, validation and testing sets above:
            input_train_classified = strcat('features25percent\classified\
                TRAINING_imageSize31_PCA', num2str(numberPCA(pca)), '_LDA', LDA,
                '_FeatureNormalize', NORM, '_features_classified.txt');
            input_test_classified = strcat('features25percent\classified\
                TEST_imageSize31_PCA', num2str(numberPCA(pca)), '_LDA', LDA,
                '_FeatureNormalize', NORM, '_features_classified.txt');
            input_other_classified = strcat('features75percent\classified\
                TEST_imageSize31_PCA', num2str(numberPCA(pca)),'_LDA', LDA,
```

```
65                  '_FeatureNormalize', NORM, '_features_classified.txt');
66              %the results files:
67              input_train_results = strcat('features25percent\results\
68                  TRAINING_imageSize31_PCA', num2str(numberPCA(pca)), '_LDA', LDA,
69                  '_FeatureNormalize', NORM, '_features_results.txt');
70              input_test_results = strcat('features25percent\results\
71                  TEST_imageSize31_PCA', num2str(numberPCA(pca)), '_LDA', LDA,
72                  '_FeatureNormalize', NORM, '_features_results.txt');
73              input_other_results = strcat('features75percent\results\
74                  TEST_imageSize31_PCA', num2str(numberPCA(pca)), '_LDA', LDA,
75                  '_FeatureNormalize', NORM, '_features_results.txt');
76
77              %run NB:
78              tic;
79              [X_test, Y_test, X_train, Y_train, bins, bin_vec] =
80                  BAYES_INPUT(input_test, input_train, input_other);
81              [prob, frac_0, frac_1] =
82                  TRAIN_BAYES(X_train, Y_train, bins, bin_vec);
83              [C_test] = TEST_BAYES(X_test, Y_test, X_train, Y_train, bins, prob,
84                  frac_0, frac_1, 1, 1, 1, 0, 0, input_test, input_test_classified,
85                  input_train, input_train_classified);
86              [acc1] = RESULTS(input_test_results, input_test_classified);
87              [acc2] = RESULTS(input_train_results, input_train_classified);
88              toc;
89
90              counter = counter + 1;
91
92          end
93      end
94  end
95  fclose('all');
```

D.2 Algorithm Input

```
1   function [X_test, Y_test, X_train, Y_train, bins, bin_vec] =
2       BAYES_INPUT(input_test, input_train, input_other)
3
4   %This program generates the required inputs for new_TRAIN_BAYES().
5   %
6   %X_train: matrix; each row consists of the features of one object in the
7   %   training set.
8   %Y_train: vector; each row contains the known class of the instance in the
9   %   corresponding rows in X_train.
10  %X_test: matrix; each row consists of the features of one object in the
```

```
%   test/validation set.
%Y_test: vector; each row contains the known class of the instance in the
%   corresponding rows in X_test.
%-------------------------------------------------------------------------
%-------------------------------------------------------------------------


%Read files into matrices
test = csvread(input_test, 0, 0);
train = csvread(input_train, 0, 0);
other = csvread(input_other, 0, 0);

stest = size(test);

X_test = test(:, 2:stest(2)-3);
Y_test = test(:, stest(2));
X_train = train(:, 2:stest(2)-3);
Y_train = train(:, stest(2));
X_other = other(:, 2:stest(2)-3);

%calculate bins
[bins, bin_vec] = BINS(X_train, X_test, X_other);

fclose('all');
```

D.3 Calculating the Bins

```
function [bins, bin_vec] = BINS(X_train, X_test, X_other)

%This program calculates the number of bins and the bin separation values
%(edge-values) for each feature.
%
% bin_vec: matrix; contains binning information, see comments in code below.
% bins: matrix; contains the edge-values of the bins of the various different
%   features, see comments in code below.
%-------------------------------------------------------------------------
%-------------------------------------------------------------------------

sx = size(X_train);
amount_of_features = sx(2);
data_points = sx(1);

%create a matrix (bin_vec) to store the amount of bins required for each
%feature in the dataset, as well as each feature's maximum and minimum
```

```
%value (across the whole dataset).
%rows = different features
%col1 = number of bins for feature
%col2 = feature maximum value
%col3 = feature minimum value
bin_vec = zeros(amount_of_features, 3);

%fill bin_vec with values:
for u = 1:amount_of_features

    %calculate the number of bins (same for every feature):
    numberOfBins = data_points / 2;
    numberOfBins = numberOfBins / 4; %average of roughly 4 objects per bin.
    numberOfBins = round(numberOfBins);

    %get the maximum and minimum values of this feature in the
    %training, validation and test set.
    maxf = zeros(3,1);
    minf = zeros(3,1);
    maxf(1,1) = max(max(X_train(:,u)));
    maxf(2,1) = max(max(X_test(:,u)));
    maxf(3,1) = max(max(X_other(:,u)));
    minf(1,1) = min(min(X_train(:,u)));
    minf(2,1) = min(min(X_test(:,u)));
    minf(3,1) = min(min(X_other(:,u)));
    %the absolute maximum and minimum for this feature for the
    %whole dataset is then:
    maxi = max(max(maxf(:,1)));
    mini = min(min(minf(:,1)));

    %store the number of bins, the maximum and the minimum:
    bin_vec(u,1) = numberOfBins;
    bin_vec(u,2) = maxi;
    bin_vec(u,3) = mini;
end

%determine the maximum amount of bins used for any feature (although this seems
%redundant in the case here where all features have an equal number of bins,
%this code makes allowance for the case where different features have
%different numbers of bins.)
max_bins = max(max(bin_vec(:,1)));
max_bins = max_bins(1);
%a = number of bin edges
a = max_bins + 1;

%create the empty "bins" matrix: the edge-values of the bins for each feature will
%be stored in it.
bins = zeros(amount_of_features, a);
```

```
66  sib = size(bins);
67
68  %store the edge-values in "bins" for each feature
69  for y = 1:amount_of_features
70
71      %get the maximum and minimum value of the feature
72      maxi = bin_vec(y,2);
73      mini = bin_vec(y,3);
74
75      %determine step size
76      step = (maxi-mini)/bin_vec(y,1);
77
78      %do the edges
79      bins(y,1) = mini;
80      val = mini;
81      for t = 2:(bin_vec(y,1)+1)
82          if t ~= bin_vec(y,1)+1
83              val = val + step;
84              bins(y,t) = val;
85          else
86              val = maxi + 0.00000000001;
87              bins(y,t) = val;
88          end
89      end
90      sub = sib(2) - (bin_vec(y,1)+1);
91      if sub ~= 0
92          for e = 1:sub
93              bins(y, sib(2)-sub+e) = val;
94          end
95      end
96  end
```

D.4 Training Naïve Bayes

```
1   function [prob, frac_0, frac_1] = TRAIN_BAYES(X_train, Y_train, bins, bin_vec)
2
3   %The goal of this program is to create a 3D matrix containing the conditional
4   %probabilities (obtained from frequency counts using the training data) of objects
5   %(given their class) having certain feature values.
6   %
7   %prob: matrix; used for storing the conditional probabilities mentioned above.
8   %frac_0: prior probaility of class "0".
9   %frac_1: prior probability of class "1".
10  %-------------------------------------------------------------------------
```

```
%-------------------------------------------------------------------------

s = size(X_train);
total = s(1);
a = size(bins);
sy = size(Y_train);

%determine the number of different classes (2 in our case: "1" and "0")
Y_u = unique(Y_train);
number_of_classes = length(Y_u);

%create empty "countj" 3D matrix for storing frequency counts
countj = zeros(s(2),a(2)-1,number_of_classes);

%create fraction variables (prior probabilities for class "0" and class "1")
frac_0 = 0;
frac_1 = 0;

%obtain the "countj" matrix
for i=1:number_of_classes

    indi = find(Y_train == Y_u(i));
    datai = X_train(indi,:);
    sdatai = size(datai);

    %determine the fractions of each class out of the total
    if Y_u(i) == 0
        frac_0 = sdatai(1)/total;
    elseif Y_u(i) == 1
        frac_1 = sdatai(1)/total;
    end

    for j = 1:s(2)      %loop through all the features
        validEdges = bin_vec(j,1)+1;
        featurejBinned = histc(datai(:,j), bins(j,1:validEdges));
        featurejBinned = featurejBinned(1:bin_vec(j,1));
        rest = a(2)-1-bin_vec(j,1);
        if rest ~= 0
            for zer = (bin_vec(j,1)+1):(a(2)-1)
                featurejBinned(zer) = 0.0;
            end
        end
        countj(j,:,i) =  featurejBinned;   %vector of counts for feature j
    end
end

%create the probability matrix
prob = zeros(s(2),a(2)-1,number_of_classes);
```

```
59
60 %obtain the probability matrix
61 sc = size(countj);
62 for i = 1:sc(1) %loop through all the features
63     validBinsForThisFeature = bin_vec(i,1);
64     for j = 1:validBinsForThisFeature %loop through the valid bins
65         for k = 1:sc(3) %loop through all the classes
66             %includes the use of Laplace smoothing:
67             prob(i,j,k) = (1 + countj(i,j,k))/(bin_vec(i,1) + sum(countj(i,:,k)));
68         end
69     end
70 end
```

D.5 Testing Naïve Bayes

```
1  function [C_test] = TEST_BAYES(X_test, Y_test, X_train, Y_train, bins, prob,
2      frac_0, frac_1, select_test, select_train, output, threshold, thresholdValue,
3      input_test, input_test_classified, input_train, input_train_classified)
4
5  %The purpose of this function is to assign a class to each of the objects
6  %in the rows of both X_test and X_train. This will then be used to output
7  %_classified.txt files, from where RESULTS.m can be used to calculate the
8  %performance metrics.
9  %
10 %C_test: vector; each row contains the predicted class of the corresponding test
11 %object in X_test.
12 %If select_test == 1, classify the test set, otherwise don't.
13 %If select_train == 1, classify the training set, otherwise don't.
14 %If output == 1, create the '_classified.txt' files, otherwise don't.
15 %If threshold == 1, take the prior probabilities of the classes into account, else
16 %   don't.(This is not valid anymore - ensure that it is always 0).
17 %Set thresholdValue to 0 for standard NB where an object belongs to the class
18 %resulting in the highest probability P(Class|object_features).
19 %-------------------------------------------------------------------------------
20 %-------------------------------------------------------------------------------
21
22 %classify test set.
23 %-------------------
24 if select_test == 1
25
26     stest = size(X_test);
27     sb = size(bins);
28     C_test = zeros(stest(1), 1);
29
```

```
30      %determine the number of different classes
31      Y_test_small = unique(Y_test);   % 0, 1
32      number_of_classes = length(Y_test_small);
33
34      %classify the data points
35      for i = 1:stest(1)    %loop through data points
36          class_vector_test = zeros(number_of_classes, 1);
37          for j = 1:number_of_classes     %loop through classes
38              class_prob = 1.0;
39              for k = 1:stest(2)     %loop through features
40                  %determine the bin's probability index that you need
41                  feature_value = X_test(i,k);
42                  gotprob = 0;
43                  for t = 1:sb(2)-1   %loop through the bins
44                      if (feature_value >= bins(k,t)) && (feature_value < bins(k, t+1))
45                          class_prob = class_prob * prob(k, t, j);
46                          gotprob = 1;
47                          break;
48                      end
49                  end
50              end
51
52              %find the right class fraction (prior probability) to multiply with.
53              if Y_test_small(j) == 0
54                  class_prob = class_prob * frac_0;
55              elseif Y_test_small(j) == 1
56                  class_prob = class_prob * frac_1;
57              end
58
59              class_vector_test(j, 1) = class_prob;
60          end
61
62          %working with the object and its probabilities now:
63          if threshold == 0
64              maxclass = find(class_vector_test == max(max(class_vector_test)));
65              if length(maxclass) > 1
66                  maxclass = maxclass(2);
67              end
68              C_test(i, 1) = maxclass - 1;
69              Pred_Prob(i,1) = (class_vector_test(2,1) / (class_vector_test(2,1) + class_vector_t
70          else
71              class_0_prob = class_vector_test(1,1);
72              class_1_prob = class_vector_test(2,1);
73              ratio = class_1_prob / class_0_prob;
74
75              if ratio >= thresholdValue
76                  %then real wins
77                  C_test(i,1) = 1;
```

```
78              else
79                  %then not-real wins
80                  C_test(i,1) = 0;
81              end
82          end
83      end
84  end
85
86  %classify training set.
87  %-----------------------
88
89  if select_train == 1
90
91      strain = size(X_train);
92      sb = size(bins);
93      C_train = zeros(strain(1), 1);
94
95      %determine the number of different classes
96      Y_train_small = unique(Y_train);   % 0, 1
97      number_of_classes = length(Y_train_small);
98
99      %classify the data points
100     for i = 1:strain(1)    %loop through data points
101         class_vector_train = zeros(number_of_classes, 1);
102         for j = 1:number_of_classes     %loop through classes
103             class_prob = 1;
104             for k = 1:strain(2)     %loop through features
105                 %determine the bin's probability index that you need
106                 feature_value = X_train(i,k);
107                 gotprob = 0;
108                 for t = 1:sb(2)-1   %loop through the bins
109                     if (feature_value >= bins(k,t)) && (feature_value < bins(k, t+1))
110                         class_prob = class_prob * prob(k, t, j);
111                         gotprob = 1;
112                         break;
113                     end
114                 end
115             end
116
117             %find the right class fraction to multiply with.
118             if Y_train_small(j) == 0
119                 class_prob = class_prob * frac_0;
120             elseif Y_train_small(j) == 1
121                 class_prob = class_prob * frac_1;
122             end
123
124             class_vector_train(j, 1) = class_prob;
125         end
```

```
126
127         if threshold == 0
128             maxclass = find(class_vector_train == max(max(class_vector_train)));
129             maxclass = maxclass(1);
130             C_train(i, 1) = maxclass - 1;
131         else
132             class_0_prob = class_vector_train(1,1);
133             class_1_prob = class_vector_train(2,1);
134             ratio = class_1_prob / class_0_prob;
135
136             if ratio >= thresholdValue
137                 %then real wins
138                 C_train(i,1) = 1;
139             else
140                 %then not-real wins
141                 C_train(i,1) = 0;
142             end
143         end
144     end
145 end
146
147
148 %now make the _classified.txt outputs.
149 %---------------------------------------
150 %---------------------------------------
151
152 if output == 1
153
154     %test-case:
155     %-----------
156
157     if select_test == 1
158
159         test_mat = csvread(input_test, 0, 0);
160         sz = size(test_mat);
161
162         test_out = fopen(input_test_classified, 'wb', 'b', 'UTF-8');
163
164         test_classified = zeros(sz(1), sz(2)+1);
165         test_classified(:, 1:sz(2)) = test_mat;
166         test_classified(:, sz(2)+1) = C_test;
167
168         sc = size(test_classified);
169
170         %print the file
171         for i = 1:sc(1)
172             for j = 1:sc(2)
173
```

```
174                 if j == 1
175                     s = sprintf('%i', test_classified(i,j));
176                     if length(s) ~= 9
177                         res = 9 - length(s);
178                         while res > 0
179                             s = strcat('0', s);
180                             res = res - 1;
181                         end
182                     end
183 
184                     if i == 1
185                         fprintf(test_out, '%s', s);
186                     else
187                         fprintf(test_out, '\n%s', s);
188                     end
189                 elseif (j == sc(2)) || (j == sc(2)-1) || (j == sc(2)-2) || (j == sc(2)-3)
190                     fprintf(test_out, ', %i', test_classified(i,j));
191                 else
192                     fprintf(test_out, ', %.5E', test_classified(i,j));
193                 end
194             end
195         end
196         fclose(test_out);
197     end
198 
199 
200     %training-case:
201     %---------------
202 
203     if select_train == 1
204 
205         train_mat = csvread(input_train, 0, 0);
206         bla = size(train_mat);
207 
208         train_out = fopen(input_train_classified, 'wb', 'b', 'UTF-8');
209 
210         train_classified = zeros(bla(1), bla(2)+1);
211         train_classified(:, 1:bla(2)) = train_mat;
212         train_classified(:, bla(2)+1) = C_train;
213 
214         sa = size(train_classified);
215 
216         %print the file
217         for i = 1:sa(1)
218             for j = 1:sa(2)
219 
220                 if j == 1
221                     s = sprintf('%i', train_classified(i,j));
```

```
                    if length(s) ~= 9
                        res = 9 - length(s);
                        while res > 0
                            s = strcat('0', s);
                            res = res - 1;
                        end
                    end

                    if i == 1
                        fprintf(train_out, '%s', s);
                    else
                        fprintf(train_out, '\n%s', s);
                    end
                elseif (j == sa(2)) || (j == sa(2)-1) || (j == sa(2)-2) || (j == sa(2)-3)
                    fprintf(train_out, ', %i', train_classified(i,j));
                else
                    fprintf(train_out, ', %.5E', train_classified(i,j));
                end
            end
        end
        fclose(train_out);
    end
end
fclose('all');
```

D.6 Calculating Performance Metrics

```
function [acc] = RESULTS(test_results, test_classified)

%This function reads in a text-file with the classified data and determines
%the performance metrics based on that. It also writes the results in an
%output file.
%-------------------------------------------------------------------------------
%-------------------------------------------------------------------------------

%create the output file for the results.
results_out = fopen(test_results, 'wb', 'b', 'UTF-8');

%load the classified data text file into a matrix.
classified_data = csvread(test_classified, 0, 0);

%get the known and predicted classification.
sd = size(classified_data);
known = classified_data(:, sd(2)-1);
```

```
18 classified = classified_data(:, sd(2));
19
20 %determine the confusion-matrix and calculate the performance metrics.
21 [mat, order] = confusionmat(known, classified);
22 acc = ((mat(1,1) + mat(2,2)) / (mat(1,1) + mat(1,2) + mat(2,1) + mat(2,2)));
23 pre = mat(2,2) / (mat(2,2) + mat(1,2));
24 rec = mat(2,2) / (mat(2,2) + mat(2,1));
25 f1 = (2 * rec * pre) / (rec + pre);
26
27 %print the output file.
28 fprintf(results_out, 'acc,  pre,  rec,  f1, mat(1,1), mat(1,2), mat(2,1), mat(2,2),
29     order(1), order(2)');
30 fprintf(results_out, '\n%.2f, %.2f, %.2f, %.2f, %d, %d, %d, %d, %i, %i', acc, pre,
31     rec, f1, mat(1,1), mat(1,2), mat(2,1), mat(2,2), order(1), order(2));
32
33 fclose('all');
34 end
```

Bibliography

[1] L. du Buisson, N. Sivanandam, B. A. Bassett, and M. Smith. Machine Learning Classification of SDSS Transient Survey Images. *ArXiv e-prints*, July 2014. URL http://arxiv.org/abs/1407.4118.

[2] N. J. Nilsson. *The Quest for Artificial Intelligence: A History of Ideas and Achievements.* Cambridge University Press, New York, NY, 2010.

[3] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, Inc., Upper Saddle River, NJ, 3nd edition, 2010.

[4] W. S. McCulloch and W. H. Pitts. A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[5] D. O. Hebb. *The Organisation of Behaviour.* John Wiley & Sons Inc., 1949.

[6] M. L. Minsky. A Neural-Analogue Calculator based upon a Probability Model of Reinforcement. Technical Report, Harvard University Psychological Laboratories, Cambridge, MA, January 1952.

[7] M. L. Minsky. *Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain-Model Problem.* PhD thesis, University of Princeton, Princeton, NJ, 1954.

[8] A. M. Turing. Computing Machinery and Intelligence. *Mind*, 59(236):433–460, October 1950.

[9] J. McCarthy, M. L. Minsky, N. Rochester, and C. B. Shannon. Proposal for the Dartmouth Summer Research Project for Artificial Intelligence. Technical Report, Dartmouth College, Hanover, NH, 1955.

[10] A Newell and H. A. Simon. The Logic Theory Machine – A Complex Information Processing System. *IRE Transactions on Information Theory*, 2(3):61–79, September 1956.

[11] B. Russell and A. N. Whitehead. *Principia Mathematica.* Cambridge University Press, 2nd edition, 1925.

[12] A. Newell and H. A. Simon. GPS, a Program that Simulates Human Thought. In H. Billing, editor, *Lerende Automaten*, pages 109–124. R. Oldenbourg, 1961.

[13] A. Newell and H. A. Simon. Computer Science as Empirical Inquiry: Symbols and Search. *Communications of the ACM*, 19(3):113–126, March 1976.

[14] A. Newell. Physical Symbol Systems. *Cognitive Science*, 4(2):135–183, April 1980.

[15] H. Gelernter. Realization of a geometry theorem proving machine. In *IFIP Congress*, pages 273–281, 1959.

[16] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959.

[17] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. II–Recent Progress. *IBM Journal of Research and Development*, 11(6):601–617, November 1967.

[18] J. McCarthy. Programs with Common Sense. In M. L. Minsky, editor, *Semantic Information Processing*, pages 403–418. MIT Press, 1968.

[19] D. Huffman. Impossible Objects as Nonsense Sentences. In B. Meltzer and D. Michie, editors, *Machine Intelligence 6*. Edinburgh University Press, 1971.

[20] T. Winograd. Understanding Natural Language. *Cognitive Psychology*, 3(1):1–191, January 1972.

[21] S. Winograd and J. D. Cowan. *Reliable Computation in the Presence of Noise*. MIT Press, Cambridge, MA, 1963.

[22] B. Widrow and M. J. Hoff. Adaptive Switching Circuits. In *IRE WESCON Convention Record*, pages 96–104, 1960.

[23] B. Widrow. Generalization and Information Storage in Networks of Adeline Neurons. In M. C. Yovitz, G. T. Jacobi, and G. D. Goldstein, editors, *Self-Organizing Systems*, pages 435–461, Chicago, 1962.

[24] F. Rosenblatt. *Principles of Neurodynamics; Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, 1962.

[25] H. D. Block, B. W. Knight, and F. Rosenblatt. Analysis of a Four-Layer Series-Coupled Perceptron. II. *Rev. Mod. Phys.*, 34(1):135–142, January 1962.

[26] R. M. Friedberg. A Learning Machine: Part I. *IBM Journal of Research and Development*, 2(1):2–13, January 1958.

[27] R. M. Friedberg, B. Dunham, and J. H. North. A Learning Machine: Part II. *IBM Journal of Research and Development*, 3(3):282–287, July 1959.

[28] M. L. Minsky and S. Papert. *Perceptrons; An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.

[29] A. E. Bryson and Y. C. Ho. *Applied Optimal Control: Optimization, Estimation, and Control*. Blaisdell, 1969.

[30] B. Buchanan, G. Sutherland, and E. A. Feigenbaum. Heuristic DENDRAL: a Program for Generating Explanatory Hypotheses in Organic Chemistry. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 209–254. Edinburgh University Press, 1969.

[31] E. H. Shortliffe. *Computer-Based Medical Consultations: MYCIN*. Elsevier B.V., 1976.

[32] B. G. Buchanan and E. H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Boston, MA, 1984.

[33] R. C. Schank and R. P. Abelson. *Scripts, Plans, Goals and Understanding: an Inquiry into Human Knowledge Structures*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.

[34] R. Wilensky. *Understanding goal-based stories*. PhD thesis, Yale University, New Haven, CT, 1978.

[35] R. C. Schank and C. K. Riesbeck. *Inside Computer Understanding: Five Programs Plus Miniatures*. Lawrence Erlbaum Associates, Hillsdale, NJ, January 1981.

[36] M. G. Dyer. *In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension*. MIT Press, Cambridge, MA, 1983.

[37] P. Langley. The Changing Science of Machine Learning. *Machine Learning*, 82:275–279, 2011.

[38] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*. MIT Press, 1986.

[39] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian Approach to Filtering Junk E-Mail. In *Learning for Text Categorization: Papers from the 1998 Workshop. AAAI Technical Report WS-98-05*, Madison, WI, July 1998.

[40] J. Goodman and D. Heckerman. Fighting Spam with Statistics. *Significance: the Magazine of the Royal Statistical Society*, 1(2):69–72, June 2004.

[41] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, et al. Stanley: The Robot that Won the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9):661–692, September 2006.

[42] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, et al. Autonomous Driving in Urban Environments: Boss and the Urban Challenge. *Journal of Field Robotics*, 25(8):425–466, August 2008.

[43] D. Goodman, R. Keene, and . *Man Versus Machine: Kasparov Versus Deep Blue*. H3 Publications, 1997.

[44] N. M. Ball and R. J. Brunner. Data Mining and Machine Learning in Astronomy. *International Journal of Modern Physics D*, 19:1049–1106, 2010.

[45] S. C. Odewahn, E. B. Stockwell, R. L. Pennington, R. M. Humphreys, and W. A. Zumach. Automated Star/Galaxy Discrimination with Neural Networks. *AJ*, 103:318–331, January 1992. doi: 10.1086/116063.

[46] S. C. Odewahn and M. L. Nielsen. Star-Galaxy Separation Using Neural Networks. *Vistas in Astronomy*, 38:281–286, 1994. doi: 10.1016/0083-6656(94)90038-8.

[47] D. Bazell and Y. Peng. A Comparison of Neural Network Algorithms and Preprocessing Methods for Star-Galaxy Discrimination. *ApJS*, 116:47–55, May 1998. doi: 10.1086/313098.

[48] S. Andreon, G. Gargiulo, G. Longo, R. Tagliaferri, and N. Capuano. Wide field imaging - I. Applications of neural networks to object detection and star/galaxy classification. *MNRAS*, 319:700–716, December 2000. doi: 10.1046/j.1365-8711.2000.03700.x.

[49] N. S. Philip, Y. Wadadekar, A. Kembhavi, and K. B. Joseph. A difference boosting neural network for automated star-galaxy classification. *A&A*, 385:1119–1126, April 2002. doi: 10.1051/0004-6361:20020219.

[50] S. C. Odewahn, R. R. de Carvalho, R. R. Gal, S. G. Djorgovski, R. Brunner, A. Mahabal, P. A. A. Lopes, J. L. K. Moreira, and B. Stalder. The Digitized Second Palomar Observatory Sky Survey (DPOSS). III. Star-Galaxy Separation. *AJ*, 128:3092–3107, December 2004. doi: 10.1086/425525.

[51] A. Collister, O. Lahav, C. Blake, R. Cannon, S. Croom, M. Drinkwater, A. Edge, D. Eisenstein, et al. MegaZ-LRG: A Photometric Redshift Catalogue of One Million SDSS Luminous Red Galaxies. *MNRAS*, 375:68–76, February 2007.

[52] N. Weir, U. M. Fayyad, and S. Djorgovski. Automated Star/Galaxy Classification for Digitized Poss-II. *AJ*, 109:2401, June 1995. doi: 10.1086/117459.

[53] N. M. Ball, R. J. Brunner, A. D. Myers, and D. Tcheng. Robust Machine Learning Applied to Astronomical Data Sets. I. Star-Galaxy Classification of the Sloan Digital Sky Survey DR3 Using Decision Trees. *ApJ*, 650:497–509, October 2006.

[54] M. C. Storrie-Lombardi, O. Lahav, L. Sodr, Jr, and L. J. Storrie-Lombardi. Morphological Classification of Galaxies by Artificial Neural Networks. *MNRAS*, 259:8P–12P, November 1992.

[55] O. Lahav, A. Naim, R. J. Buta, H. G. Corwin, G. de Vaucouleurs, A. Dressler, J. P. Huchra, S. van den Bergh, et al. Galaxies, Human Eyes, and Artificial Neural Networks. *Science*, 267:859–862, February 1995. doi: 10.1126/science.267.5199.859.

[56] A. Naim, O. Lahav, R. J. Buta, H. G. Corwin, Jr, A. Dressler, J. P. Huchra, S. van den Bergh, S. Raychaudhury, et al. A Comparative Study of Morphological Classifications of APM Galaxies. *MNRAS*, 274:1107–1125, June 1995.

[57] A. Naim, O. Lahav, L. Sodre, Jr., and M. C. Storrie-Lombardi. Automated Morphological Classification of APM Galaxies by Supervised Artificial Neural Networks. *MNRAS*, 275: 567–590, August 1995.

[58] O. Lahav, A. Naim, L. Sodré, Jr., and M. C. Storrie-Lombardi. Neural Computation as a Tool for Galaxy Classification: Methods and Examples. *MNRAS*, 283:207, November 1996.

[59] A. A. Collister and O. Lahav. ANNz: Estimating Photometric Redshifts Using Artificial Neural Networks. *PASP*, 116:345–351, April 2004. doi: 10.1086/383254.

[60] A. Naim, K. U. Ratnatunga, and R. E. Griffiths. Quantitative Morphology of Moderate-Redshift Galaxies: How Many Peculiar Galaxies Are There? *ApJ*, 476:510–520, February 1997.

[61] D. S. Madgwick. Correlating Galaxy Morphologies and Spectra in the 2dF Galaxy Redshift Survey. *MNRAS*, 338:197–207, January 2003. doi: 10.1046/j.1365-8711.2003.06033.x.

[62] S. C. Odewahn, R. A. Windhorst, S. P. Driver, and W. C. Keel. Automated Morphological Classification in Deep Hubble Space Telescope UBVI Fields: Rapidly and Passively Evolving Faint Galaxy Populations. *ApJL*, 472:L13–L16, November 1996.

[63] R. Windhorst, S. Odewahn, C. Burg, S. Cohen, and I. Waddington. Young and Old Galaxies at High Redshift. *Ap&SS*, 269:243–262, December 1999. doi: 10.1023/A:1017059922502.

[64] S. H. Cohen, R. A. Windhorst, S. C. Odewahn, C. A. Chiarenza, and S. P. Driver. The Hubble Space Telescope WFPC2 B-Band Parallel Survey: A Study of Galaxy Morphology for Magnitudes 18≤B≤27. *AJ*, 125:1762–1783, April 2003. doi: 10.1086/368367.

[65] A. J. Connolly, A. S. Szalay, M. A. Bershady, A. L. Kinney, and D. Calzetti. Spectral Classification of Galaxies: an Orthogonal Approach. *AJ*, 110:1071, September 1995.

[66] A. J. Connolly and A. S. Szalay. A Robust Classification of Galaxy Spectra: Dealing with Noisy and Incomplete Data. *AJ*, 117:2052–2062, May 1999.

[67] D. Madgwick, O. Lahav, K. Taylor, and 2dFGRS Team. Parameterisation of Galaxy Spectra in the 2dF Galaxy Redshift Survey. In A. J. Banday, S. Zaroubi, and M. Bartelmann, editors, *Mining the Sky*, page 331, 2001. doi: 10.1007/10849171_39.

[68] C. W. Yip, A. J. Connolly, A. S. Szalay, T. Budavári, M. SubbaRao, J. A. Frieman, R. C. Nichol, A. M. Hopkins, et al. Distributions of Galaxy Spectral Types in the Sloan Digital Sky Survey. *AJ*, 128:585–609, August 2004.

[69] R. Carballo, A. S. Cofiño, and J. I. González-Serrano. Selection of Quasar Candidates from Combined Radio and Optical Surveys Using Neural Networks. *MNRAS*, 353:211–220, September 2004.

[70] J.-F. Claeskens, A. Smette, L. Vandenbulcke, and J. Surdej. Identification and Redshift Determination of Quasi-Stellar Objects with Medium-Band Photometry: Application to Gaia. *MNRAS*, 367:879–904, April 2006.

[71] R. Carballo, J. I. González-Serrano, C. R. Benn, and F. Jiménez-Luján. Use of Neural Networks for the Identification of New z $\geq$ 3.6 QSOs from FIRST-SDSS DR5. *MNRAS*, 391: 369–382, November 2008.

[72] R. L. White et al. The FIRST Bright Quasar Survey. II. 60 Nights and 1200 Spectra Later. *ApJS*, 126:133–207, February 2000.

[73] Y. Zhang and Y. Zhao. A Comparison of BBN, ADTree and MLP in Separating Quasars from Large Survey Catalogues. *Chinese Journal of Astronomy and Astrophysics*, 7:289, 2007.

[74] C. Knigge, S. Scaringi, M. R. Goad, and C. E. Cottis. The Intrinsic Fraction of Broad-absorption Line Quasars. *MNRAS*, 386:1426–1435, May 2008.

[75] M. Zhao, C. Wu, A. Luo, F. Wu, and Y. Zhao. Automated Spectral Classification of Broad-line and Narrow-line Active Galactic Nuclei Based on the K-nearest Neighbor Method. *Chinese Astronomy and Astrophysics*, 31:352–362, 2007.

[76] A. E. Firth, O. Lahav, and R. S. Somerville. Estimating Photometric Redshifts with Artificial Neural Networks. *MNRAS*, 339:1195–1202, March 2003.

[77] N. M. Ball, J. Loveday, M. Fukugita, O. Nakamura, S. Okamura, J. Brinkmann, and R. J. Brunner. Galaxy Types in the Sloan Digital Sky Survey using Supervised Artificial Neural Networks. *MNRAS*, 348:1038–1046, March 2004.

[78] L. Li, Y. Zhang, Y. Zhao, and D. Yang. Estimating Photometric Redshifts with Artificial Neural Networks and Multi-Parameters. *Chinese Journal of Astronomy and Astrophysics*, 7: 448, 2006.

[79] M. Banerji, F. B. Abdalla, O. Lahav, and H. Lin. Photometric Redshifts for the Dark Energy Survey and VISTA and Implications for Large-scale Structure. *MNRAS*, 386:1219–1233, May 2008.

[80] Y. Zhang, L. Li, and Y. Zhao. Morphology Classification and Photometric Redshift Measurement of Galaxies. *MNRAS*, 392:233–239, January 2009.

[81] Y. Wadadekar. Estimating Photometric Redshifts Using Support Vector Machines. *PASP*, 117:79–85, January 2005.

[82] D. Wang, Y. Zhang, C. Liu, and Y. Zhao. Two Novel Approaches for Photometric Redshift Estimation based on SDSS and 2MASS. *Chinese Journal of Astronomy and Astrophysics*, 8 (1):119, 2008.

[83] S. Carliles, T. Budavári, S. Heinis, C. Priebe, and A. Szalay. Photometric Redshift Estimation on SDSS Data Using Random Forests. In R. W. Argyle, P. S. Bunclark, and J. R. Lewis, editors, *Astronomical Data Analysis Software and Systems XVII*, volume 394 of *Astronomical Society of the Pacific Conference Series*, page 521, August 2008.

[84] N. M. Ball, R. J. Brunner, A. D. Myers, N. E. Strand, S. L. Alberts, and D. Tcheng. Robust Machine Learning Applied to Astronomical Data Sets. III. Probabilistic Photometric Redshifts for Galaxies and Quasars in the SDSS and GALEX. *ApJ*, 683:12–21, August 2008.

[85] N. M. Ball, R. J. Brunner, A. D. Myers, N. E. Strand, S. L. Alberts, D. Tcheng, and X. Llorà. Robust Machine Learning Applied to Astronomical Data Sets. II. Quantifying Photometric Redshifts for Quasars Using Instance-based Learning. *ApJ*, 663:774–780, July 2007.

[86] J. F. Ramírez, O. Fuentes, and R. K. Gulati. Prediction of Stellar Atmospheric Parameters using Instance-Based Machine Learning and Genetic Algorithms. *Experimental Astronomy*, 12:163–178, 2001.

[87] M. R. Mokiem, A. de Koter, J. Puls, A. Herrero, F. Najarro, and M. R. Villamariz. Spectral Analysis of Early-Type Stars Using a Genetic Algorithm Based Fitting Method. *A& A*, 441: 711–733, October 2005.

[88] P. Graff, F. Feroz, M. P. Hobson, and A. Lasenby. SKYNET: an efficient and robust neural network training tool for machine learning in astronomy. *MNRAS*, 441:1741–1759, June 2014. doi: 10.1093/mnras/stu642. URL http://arxiv.org/abs/1309.0790.

[89] E. Bertin and S. Arnouts. SExtractor: Software for Source Extraction. *A& AS*, 117:393–404, June 1996.

[90] S. Giridhar, A. Goswami, A. Kunder, S. Muneer, and G. Selvakumar. Identification of Metal-poor Stars Using the Artificial Neural Network. *A& A*, 556:A121, August 2013.

[91] A. C. Becker. Transient Detection and Classification. *Astronomische Nachrichten*, 329:280, March 2008.

[92] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, 2006.

[93] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, NY, international edition, 1997.

[94] R. E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[95] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, NY, 2nd edition, 2009.

[96] G. Hughes. On the Mean Accuracy of Statistical Pattern Recognizers. *IEEE Transactions on Information Theory*, 14(1):55–63, January 1968.

[97] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[98] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517, September 1975. ISSN 0001-0782. doi: 10.1145/361002.361007. URL http://doi.acm.org/10.1145/361002.361007.

[99] J. Friedman, J. Bentley, and R. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[100] M. A. Aizerman, E. M. Braverman, and L. I. Rozonoer. The probability problem of pattern recognition learning and the method of potential functions. *Automation and Remote Control*, 25:1175–1190, 1964.

[101] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the Fifth Annual Workshop on Computational Learning Theory (COLT)*, pages 144–152. ACM, 1992.

[102] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[103] K. P. Bennett. Robust Linear Programming Discrimination of Two Linearly Separable Sets. *Optimization Methods and Software*, 1:23–34, 1992.

[104] C. Cortes and V. N. Vapnik. Support Vector Networks. *Machine Learning*, 20:273–297, 1995.

[105] J. C. Platt. Probabilities for SV Machines. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Shuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–73. MIT Press, 2000.

[106] K. Hornik, M. Stinchcombe, and H. White. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2(5):359–366, 1989.

[107] S. Geva and J. Sitte. A Constructive Method for Multivariate Function Approximation by Multilayer Perceptrons . *IEEE Transactions on Neural Networks*, 3(4):621–624, 1992.

[108] F. Murtagh. Multilayer Perceptrons for Classification and Regression. *International Journal of Neurocomputing*, 2:183–197, 1991.

[109] M. Serra-Ricart, X. Calbet, L. Garrido, and V. Gaitan. Multidimensional Statistical Analysis Using Artificial Neural Networks - Astronomical Applications. *AJ*, 106:1685–1695, 1993.

[110] T. D. Sanger. Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network. *Neural Networks*, 2(6):459–473, 1989.

[111] I. T. Jolliffe. *Principal Component Analysis*. Springer, New York, NY, 2nd edition, 2002.

[112] K. Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science, Sixth Series*, 2:559–572, 1901.

[113] H. Hotelling. Analysis of a complex of statistical variables into principal components. *The Journal of Educational Psychology*, 24:417–441, 1933.

[114] R. A. Fisher. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7:179–188, 1936.

[115] W. Baade and F. Zwicky. Remarks on Super-Novae and Cosmic Rays. *Physical Review*, 46: 76, July 1934.

[116] K. Lundmark. Nebulæ, The motions and the distances of spiral. *MNRAS*, 85:865, June 1925.

[117] A. V. Filippenko. Optical Spectra of Supernovae. *Annual Review of Astronomy and Astrophysics*, 35:309–355, September 1997.

[118] A. Goobar and B. Leibundgut. Supernova Cosmology: Legacy and Future. *Annual Review of Nuclear and Particle Science*, 61:251–279, February 2011.

[119] G. G. Raffelt. Neutrinos and the Stars. In *Proceedings ISAPP School*, 2011. URL http://arxiv.org/abs/1201.1637.

[120] D. Maoz, F. Mannucci, and G. Nelemans. Observational Clues to the Progenitors of Type Ia Supernovae. *Annual Review of Astronomy and Astrophysics*, 52:107–170, August 2014.

[121] A. Heger, C. L. Fryer, S. E. Woosley, N. Langer, and D. H. Hartmann. How Massive Single Stars End Their Life. *ApJ*, 591:288–300, July 2003. doi: 10.1086/375341.

[122] M. Dan, S. Rosswog, J. Guillochon, and E. Ramirez-Ruiz. How the merger of two white dwarfs depends on their mass ratio: orbital stability and detonations at contact. *MNRAS*, 422:2417–2428, May 2012.

[123] A. Gal-Yam. Luminous Supernovae. *Science*, 337:927–, August 2012. doi: 10.1126/science.1203601.

[124] Jerusalem Winter School for Theoretical Physics. Supernovae. Volume 6, Jerusalem, Dec. 28, 1988- Jan. 5, 1989. In J. C. Wheeler, T. Piran, and S. Weinberg, editors, *Supernovae, Jerusalem Winter School for Theoretical Physics*, 1990.

[125] J. C. Wheeler and R. P. Harkness. Type i supernovae. *Reports on Progress in Physics*, 53 (12):1467, 1990. URL http://stacks.iop.org/0034-4885/53/i=12/a=001.

[126] R. A. Chevalier. Hydrodynamic Models of Supernova Explosions. *Fundamentals of Cosmic Physics*, 7(1):1–58, January 1981.

[127] J. B. Doggett and D. Branch. A Comparative Study of Supernova Light Curves. *ApJ*, 90: 2303–2311, November 1985. doi: 10.1086/113934.

[128] R. P. Kirshner. *Supernovae*. Springer-Verlag, New York, 1990.

[129] T. R. Young and D. Branch. Absolute Light Curves of Type II Supernovae. *ApJL*, 342: L79–L82, July 1989. doi: 10.1086/185489.

[130] A. Kim. Stretched and Non-Stretched B-Band Supernova Light Curves. *LBNL Report LBNL-56164*, 2008.

[131] M. M. Phillips. The Absolute Magnitudes of Type IA Supernovae. *ApJl*, 413:L105–L108, August 1993. doi: 10.1086/186970.

[132] A. G. Riess, W. H. Press, and R. P. Kirshner. A Precise Distance Indicator: Type IA Supernova Multicolor Light-Curve Shapes. *ApJ*, 473:88, December 1996.

[133] R. Tripp. A two-parameter luminosity correction for Type IA supernovae. *Astronomy and Astrophysics*, 331:815–820, March 1998.

[134] S. Perlmutter, G. Aldering, and G. Goldhaber. Measurements of ω and λ from 42 high-redshift supernovae. *ApJ*, 517:565–586, June 1999.

[135] A. G Riess, A. V. Filippenko, and P. Challis. Observational Evidence from Supernovae for an Accelerating Universe and a Cosmological Constant. *AJ*, 116:1009–1038, September 1998.

[136] LSST Science Collaboration et al. LSST Science Book, Version 2.0. *ArXiv e-prints*, December 2009. URL http://arxiv.org/abs/0912.0201.

[137] M. Kunz, B. A. Bassett, and R. A. Hlozek. Bayesian Estimation Applied to Multiple Species. *Phys. Rev. D*, 75:103508, May 2007. doi: 10.1103/PhysRevD.75.103508. URL http://link.aps.org/doi/10.1103/PhysRevD.75.103508.

[138] H. Campbell, C. B. D'Andrea, and R. C. Nichol. Cosmology with Photometrically Classified Type Ia Supernovae from the SDSS-II Supernova Survey. *ApJ*, 763:88, January 2013.

[139] R. Hlozek, M. Kunz, B. Bassett, et al. Photometric Supernova Cosmology with BEAMS and SDSS-II. *ApJ*, 752:79, June 2012.

[140] R. Kessler, B. Bassett, P. Belov, et al. Results from the Supernova Photometric Classification Challenge. *PASP*, 122:1415–1431, December 2010.

[141] Y. Gong, A. Cooray, and X. Chen. Cosmology with Photometric Surveys of Type Ia Supernovae. *ApJ*, 709:1420, January 2010.

[142] J. Newling, B. A. Bassett, R. Hlozek, M. Kunz, M. Smith, and M. Varughese. Parameter Estimation with BEAMS in the Presence of Biases and Correlations. *ArXiv e-prints*, October 2011. URL http://arxiv.org/abs/1110.6178.

[143] M. Knights, B. A. Bassett, M. Varughese, R. Hlozek, M. Kunz, M. Smith, and J. Newling. Extending BEAMS to Incorporate Correlated Systematic Uncertainties. *JCAP*, 01:039, January 2013.

[144] A. M. Smith, S. Lynn, M. Sullivan, et al. Galaxy Zoo Supernovae. *MNRAS*, 412:1309–1319, 2011.

[145] S. Danziger, J. Levav, L. Avnaim-Pesso, et al. Extraneous Factors in Judicial Decisions. *PNAS*, 108:6889–6892, April 2011.

[146] S. Bailey, C. Aragon, R. C. Romano, R. Thomas, B. A. Weaver, and D. Wong. How to find more Supernovae with less work: Object Classification Techniques for Difference Imaging. *ApJ*, 665:1246–1253, August 2007.

[147] R. Romano, C. Aragon, and C. Ding. Supernova Recognition Using Support Vector Machines. In *Proceedings of the 5th International Conference on Machine Learning and Applications ICMLA 06*, pages 77–82. IEEE Computer Society Washington, DC, 2006.

[148] J. S. Bloom, J. W. Richards, P. J. Nugent, et al. Automating Discovery and Classification of Transients and Variable Stars in the Synoptic Survey Era. *PASP*, 124:1175–1196, October 2012.

[149] H. Brink, J. W. Richards, D. Poznanski, J. S. Bloom, J. Rice, S. Negahban, and M. Wainwright. Using Machine Learning for Discovery in Synoptic Survey Imaging Data. *MNRAS*, 435:1047–1060, October 2013.

[150] J. A. Frieman, B. Bassett, A. Becker, et al. The Sloan Digital Sky Survey-II Supernova Survey: Technical Summary. *AJ*, 135:338–347, January 2008. doi: 10.1088/0004-6256/135/1/338.

[151] J. A. Smith, D. L. Tucker, S. Kent, et al. The *u'g'r'i'z'* standard-star system. *AJ*, 123:2121–2144, April 2002.

[152] Ž. Ivezić, J. A. Smith, G. Miknaitis, et al. Sloan Digital Sky Survey Standard Star Catalog for Stripe 82: The dawn of industrial 1% optical photometry. *AJ*, 134:973–998, September 2007. doi: 10.1086/519976.

[153] J. B. Oke and J. E. Gunn. Secondary standard stars for absolute spectrophotometry. *ApJ*, 266:713–717, March 1983. doi: 10.1086/160817.

[154] S. Jha, R. P. Kirshner, P. Challis, et al. *UBVRI* Light Curves of 44 Type Ia Supernovae. *AJ*, 131:527–554, January 2006.

[155] J. S. Gallagher, P. M. Garnavich, P. Berlind, P. Challis, S. Jha, and R. P. Kirshner. Chemistry and Star Formation in the Host Galaxies of Type Ia Supernovae. *ApJ*, 634:210–226, November 2005.

[156] M. Sullivan, D. Le Borgne, C. J. Pritchet, et al. Rates and Properties of Type Ia Supernovae as a Function of Mass and Star Formation in their Host Galaxies. *ApJ*, 648:868–883, September 2006.

[157] M. Fukugita, T. Ichikawa, J. E. Gunn, M. Doi, K. Shimasaku, and D. P. Schneider. The Sloan Digital Sky Survey Photometric System. *AJ*, 111:1748–1756, April 1996. doi: 10.1086/117915.

[158] J. E. Gunn, W. A. Siegmund, E. J. Mannery, et al. The 2.5m Telescope of the Sloan Digital Sky Survey. *AJ*, 131:2332–2359, April 2006.

[159] J. E. Gunn, M. Carr, C. Rockosi, et al. The Sloan Digital Sky Survey Photometric Camera. *AJ*, 116:3040–3081, December 1998.

[160] M. Sako, B. Bassett, A. C. Becker, et al. The data release of the Sloan Digital Sky Survey-II Supernova Survey. *ArXiv e-prints*, January 2014.

[161] R. H. Lupton, J. E. Gunn, Ž. Ivezić, G. R. Knapp, S. M. Kent, and N. Yasuda. Astronomical Data Analysis Software and Systems. In F. R. Harnden, F. A. Primini, and H. E. Payne, editors, *ASP Conf. Ser. 238*, page 269. San Francisco:ASP, 2001.

[162] C. Alard and R. H. Lupton. A Method for Optimal Image Subtraction. *ApJ*, 503:325–331, August 1998.

[163] M. Sako, B. Bassett, and A. Becker. The Sloan Digital Sky Survey-II Supernova Survey: Search Algorithm and Follow-up Observations. *AJ*, 135:348–373, January 2008.

[164] K. Abazajian, J. K. Adelman-McCarthy, M. A. Agüeros, S. S. Allam, S. F. Anderson, J. Annis, N. A. Bahcall, I. K. Baldry, et al. The First Data Release of the Sloan Digital Sky Survey. *AJ*, 126:2081–2086, October 2003. doi: 10.1086/378165.

[165] B. A. Bassett, M. Richmond, and J. Frieman. SDSS-II SN Super-Unofficial Hand-Scanning Guide, 2005. URL http://spiff.rit.edu/richmond/sdss/sn_survey/scan_manual/sn_scan.html.

[166] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[167] M. E. Tipping and C. M. Bishop. Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society (Series B)*, 61(3):611–622, 1999.

[168] T. Fawcett. An Introduction to ROC Analysis. *Pattern Recognition Letters*, 27:861–874, 2006.

[169] J. A. Hanley and B. J. McNeil. The Meaning and Use of the Area under a Receiver Operating Characteristic (ROC) Curve. *Radiology*, 143:29–36, April 1982.

[170] MATLAB 7.8.0.347 (Release 2009a). The MathWorks, Inc. Natick, Massachusetts, United States.

[171] D. Freedman and P. Diaconis. On the Histogram as a Density Estimator: L_2 Theory. *Probability Theory and Related Fields*, 57(4):453–476, December 1981.

[172] J. Cohen. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20:37, 1960.

[173] J. L. Fleiss, B. Levin, and M. Cho Paik. *Statistical Methods for Rates and Proportions.* John Wiley & Sons, Inc., Hoboken, New Jersey & Canada, 3rd edition, 2003.

[174] D. G. Altman. *Practical Statistics for Medical Research.* Chapman & Hall, London, 1991.

[175] J. R. Landis and G. G. Koch. The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33:159–174, March 1977.

[176] R. Kessler. Private Communications, 2007.

[177] W. Karush. Minima of Functions of Several Variables with Inequalities as Side Constraints. Master's thesis, Department of Mathematics, University of Chicago, Chicago, IL, 1939.

[178] H. W. Kuhn and A. W. Tucker. Nonlinear Programming. In *Proceedings of the 2nd Berkeley Symposium on Mathematical Statistics and Probabilities*, pages 481–492. University of California Press, 1951.

[179] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, 1981.

[180] R. Fletcher. *Practical Methods of Optimization.* Wiley, 2nd edition, 1987.

[181] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, 1999.

[182] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989.

[183] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362. MIT Press, 1986.