MANCHESTER 1824
The University of Manchester

Building a deep generative model for surveyed pulsar classes

A thesis submitted to the University of Manchester for the degree of
Master of Philosophy
in the Faculty of Science and Engineering

2021

Tahina P. Ranaivomanana
Schools of Physics and Astronomy

Contents

Word Counts: 23961

List of Tables

List of Figures

Abstract

Pulsars are one-of-a-kind astronomical objects that are classified as highly magnetised and fast-rotating neutron stars. In addition to their extreme conditions of gravity, density, and magnetic fields, these objects are considered to be the most precise natural clocks, making them an excellent physics laboratory, notably for gravitational wave detection. Increasing the number of newly discovered pulsars will improve the precision of physical experiments and the understanding of pulsar populations. To this aim, existing pulsar surveys have made an effort to discover as many pulsars as possible, which has resulted in the generation of a large number of pulsar candidates. This amount of data will increase with the sensitivity of modern future pulsar surveys such as the Square Kilometre Array. Given the number of pulsar candidates produced by pulsar surveys, searching for pulsars is becoming a demanding task. To date, automated pulsar classification has been developed through the use of machine learning techniques to increase the detection rate of pulsars. This approach is, however, challenged by the scarcity of labelled pulsar data and the presence of huge amounts of noise or interference in the data, limiting the potential of this technique. This thesis seeks to address these issues by investigating machine learning classifiers that could improve pulsar candidate classification performance while keeping the number of false positives in the classification as small as possible.

The first part of our analyses made use of tabular data of various pulsar characteristics from the High Time Resolution Universe (HTRU) survey to train three different classifiers using eight features: Random Forest, XGBoost, and artificial neural networks. As our datasets are highly imbalanced, data sampling is applied to balance the number of pulsar candidates in our training data. The results show that XGBoost achieved an F1-score of 0.970 and a specificity of about 1.000, while Random Forest produced an F1-score of 0.954 and a specificity of 1.000. Sampling the training data improved the recall score of the classifiers and a deep study of data sampling techniques might be required to improve the F1-score and specificity along with the recall score.

The second part of our study explored semi-supervised machine learning techniques to address the lack of labelled pulsar candidates using two generative models: Bad-GAN and SGAN. These models were trained on four types of datasets: DM-curve, frequency-phase, pulse-profile, and time-phase. Our results show that the models trained on the DM-curve datasets achieved the best scores: using only 100 labelled examples in the DM-curve dataset, Bad-GAN achieved an F1-score of 0.941 and a specificity of 0.942, while SGAN produced an F1-score of 0.940 and a specificity of 0.952. Overall, the performance of SGAN is better on smaller labelled examples, which might be useful for classifying pulsars when future pulsar surveys start. On the other hand, Bad-GAN requires a sufficient number of labelled datasets to achieve good performance scores and might be more appropriate for candidate classification when more labelled datasets are collected from future pulsar surveys.

Declaration

I, Tahina Princy Ranaivomanana, hereby confirm that this dissertation titled, “Building a deep generative model for surveyed pulsar classes”, is my own original work, except where specific reference is made to the work of others, and that no portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright Statement

1. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

2. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

3. The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

4. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy[1], in any relevant Thesis restriction declarations deposited in the University Library, The University Library regulations [2] and in The University policy on Presentation of Theses

[1] documents.manchester.ac.uk
[2] www.library.manchester.ac.uk/about/regulations/

Acknowledgements

First and foremost, I want to express my gratitude to my supervisor, Ben Stappers, for his kind guidance, helpful comments, constructive suggestions, and thorough reviews, all of which significantly improved the quality of this thesis. My gratitude also goes to my co-supervisor, Zafiirah Hosenie, for her gentle help along the way and for providing such constructive comments and useful tips, which greatly aided in the completion of this thesis. I do not forget to thank Rob Lyon for his help and input into this work.

Secondly, I would like to give special thanks to Anna Scaife of the Jodrell Bank Centre for Astrophysics (JBCA) for allowing me to carry out my MPhil study at the University of Manchester through the Development in Africa with Radio Astronomy (DARA) Big Data programme and the Newton Fund project. It is a great privilege for me to be part of this programme. A big thanks to Linzi Stirrup for always being there during challenging times, who worked tirelessly so that I was able to settle in in the UK.

I also thank everyone from the pulsar groups and JBCA, in particular Emma, Avishek, Manisha, Laila, and my fellow friends from DARA: Ann, Chelmis, Kelvin, and David. It has been a pleasure to be with all of you, even for a short time.

Last but not least, I would like to express my gratitude to my parents and my brothers, Jessy, Elie, and Jerry, for their support and encouragement throughout these times.

Chapter 1

Introduction

Neutron stars have been an object of interest for many scientists since the discovery of the first radio pulsar. The first prediction that neutron stars might exist was made in 1934 by Walter Baade and Fritz Zwicky, who suggested that a supernova explosion would result in a neutron star. They proposed that this new type of star is a dense object with a small radius, and is mainly composed of neutrons (Baade & Zwicky, 1934). The discovery of Jocelyn Bell and Antony Hewish of the first known pulsar led to the confirmation that this type of star existed. Given the high stellar density and the conservation of magnetic flux, neutron star magnetic fields could increase up to 10^{16} G (Woltjer, 1964). The rotational kinetic energy of these strongly magnetised objects is the source of the emission of electromagnetic waves from the neutron stars (Pacini, 1967). It is thought that there are hundreds of thousands of neutron stars in the galaxy, but those that can be detected from their surface emission are only the young and hot neutron stars seen in X-rays or those that are accreting from a companion star. The vast majority of the neutron stars we know about are pulsars, that is, those that are generating emission through particle flow in their ultra-strong magnetic fields, these sources are seen across the electromagnetic spectrum, but the majority are seen in the radio.

1.1 Overview

1.1.1 Neutron stars

Neutron stars are the end-product of the final evolutionary stage of normal massive stars of stellar mass $M_\star \sim 8 - 20\ M_\odot$ (Lyne & Graham-Smith, 2012). The final stage of stellar evolution occurs when a star's main nuclear fuel is depleted and it eventually collapses under its gravity. Their cores shrink and become hotter and denser as a result of this event. The latter leaves behind heavier elements to form an iron core in the innermost region. Therefore, nuclear fusion ceases (as iron is very stable) and the stars are unable to provide enough energy to counterbalance the large gravitational forces in the region. This leads to a rapid gravitational collapse of the iron core of the stars, resulting in a massive explosion known

as a supernova (SN). The remnants of the SN explosion form a highly compact object. The density of the iron core is a key factor in determining whether the SN remnant will be a neutron star, a white dwarf, or a black hole (e.g Lyne & Graham-Smith 2012). In most cases, neutron stars are observed as rapidly rotating, highly magnetised objects known as *pulsars*.

1.1.2 Discovery of pulsars

Pulsars are highly-magnetised, rapidly rotating neutron stars with a light-house beam of radiation that triggers the pulsed emission from their magnetic poles. The first known pulsar was discovered in 1967 by Jocelyn Bell Burnell, a research student, and Antony Hewish, her advisor at Cambridge University. Bell Burnell was working on a large array of thousands of dipole antennae operating at 81.5 MHz for investigating interplanetary scintillation of compact radio sources (quasars). Meanwhile, she detected pulses of radiation lasting for 0.3 sec, which were seen to be separated by 1.337 sec on her chart recorder. Interestingly, the pulses came from the same part of the sky four minutes earlier each night, which takes into account sidereal time. In addition, analyses of the dispersion of the radio signal provide a rough estimation of the distance to the source of 65 pc away from the Earth. Therefore, it is most likely that these pulses came from objects outside the solar system rather than electromagnetic interference sources. Also, it was confirmed that these observed pulses originated from natural radio emission within our galaxy when other radio pulses were detected from a different location in the sky late in the same year of discovery. The nature of this radio source was referred to as a rapidly rotating neutron star by Thomas Gold and Sir Fred Hoyle soon after the official announcement of the pulsar discovery; later it was recognised as a magnetised, rotating neutron star (Shapiro & Teukolsky, 1983).

Discovery of other pulsar populations

The first discovered pulsar is designated as PSR B1919+21 located in the constellation of Vulpecula. PSR stands for Pulsating Source of Radio. The letter B refers to the Besselian coordinate system (epoch 1950), and 1919+21 corresponds to the location of the source: right ascension (RA: 19h19') and declination (DEC: +21°). Originally, this pulsar is known as CP 1919 (Cambridge Pulsar at RA 19h19min). Various types of neutron stars (Kaspi & Kramer, 2016) have been observed since then, such as radio pulsars, binary radio pulsars, magnetars, isolated neutron stars (INSs), and rotating radio transients (RRATs). Among the most interesting discoveries in pulsar astronomy is the discovery of a new class of pulsars, known as millisecond pulsars (MSPs, Lorimer 2005). It is defined as a pulsar with a short-rotational period and a small spin-down rate (see Section 1.4.1). They are beneficial not only for understanding neutron star evolution and the equation of state of condensed matter, but also for detecting low-frequency gravitational waves and testing theories of gravity (Lorimer et al., 2007).

1.2 Properties of pulsars

1.2.1 Density and radius

Neutron stars have a massive nucleus with a mean density similar to that of the nucleus of an atom. The neutron stars' density and the stability of their spin period rule out the possibility that the observed pulses are from other compact objects such as black holes and white dwarfs. Furthermore, the highest spin frequency is reached when a star rotates at a speed close to the break-up velocity. Thus, considering a rotating star with an angular velocity Ω, the relation between the period of the star and its density can be determined for a self-gravitating system by (Ghosh, 2007):

$$\Omega^2 R < \frac{GM}{R^2}, \tag{1.1}$$

where M and R are, respectively, the mass and radius of the star. With $\Omega = 2\pi/P$, this equation can be rewritten as:

$$\frac{\pi}{P^2} < G\left(\frac{M}{4\pi R^3}\right) \quad \text{or} \quad \frac{3\pi}{\mathrm{P}^2} < \mathrm{G}\left(\frac{3\mathrm{M}}{4\pi \mathrm{R}^3}\right). \tag{1.2}$$

This relation can be expressed in terms of the density of the star as:

$$\rho > \frac{3\pi}{GP^2}\ . \tag{1.3}$$

In case of the Crab pulsar with a period $P = 0.033\,s$, the former relation gives a density of about $\rho \gtrsim 10^{11}$ g/cm^3. While the fastest pulsars, known as PSR J1748−2446ad (Hessels et al., 2006), rotates at $P = 1.396\,ms$, has a density of about $\rho \gtrsim 10^{14}$ g/cm^3,which is far greater than the density of the white dwarfs. From Equation (1.2), one can derived the maximum radius of the neutron star as:

$$R < \left(\frac{GMP^2}{4\pi^2}\right)^{1/3}. \tag{1.4}$$

1.2.2 Magnetic field and spin properties

The magnetic pole of pulsars is not often aligned with its rotational axis. Pulsars can be only detected if the beam of radiation is pointed in the direction of the Earth's line of sight. Additionally, pulsars have enormous gravitational and magnetic forces (Lorimer et al., 2007), with a surface gravity approximately 10^9 times that of the Earth, and magnetic fields ranging from 10^8 to 10^{14} Gauss (Cordes et al., 2006). It is considered that the source of pulsar radio wave emission is a result of the interactions between their surfaces and their tremendous magnetic fields. Furthermore, the angular momentum conservation principle can explain the fast rotation of pulsars: as the radius of the star shrinks as a result of the supernova

explosion, it spins faster. The spin time varies from a few milliseconds to several seconds. Young pulsars tend to spin faster than old pulsars, as old pulsars lose most of their rotational energy, which results in slow rotation and large periods (Lorimer, 2008).

1.2.3 Internal structure

A neutron star is made up of two main components as illustrated in Figure 1.1: a solid crust and a liquid core (Baym & Pethick, 1979). The solid crust is a combination of an outer crust, dominated by free electrons and iron nuclei, and an inner crust, a composition of free neutrons, electrons, and heavier atomic nuclei. This solid crust is characterised by a density below $\rho_s = 2.7 \times 10^{14}$ g/cm^3, where neutrons are ideally produced; forming nuclei with high numbers of neutrons. On the other hand, the liquid core is characterised by a density above ρ_s, where heavier elements are unstable. Most protons combine with electrons to form neutron fluid. As a result, this core is formed by an ultra-dense matter, with a density of up to 10^{15} g/cm^3, and a super-fluid core of neutrons with a small fraction of electrons and protons (Lyne & Graham-Smith, 2012). The composition of the innermost core of the neutron stars, with a density of $\sim 6\rho_s$, is still a subject of research; nonetheless, a recent study suggested the evidence of exotic quark matter inside the core (Annala et al., 2020).

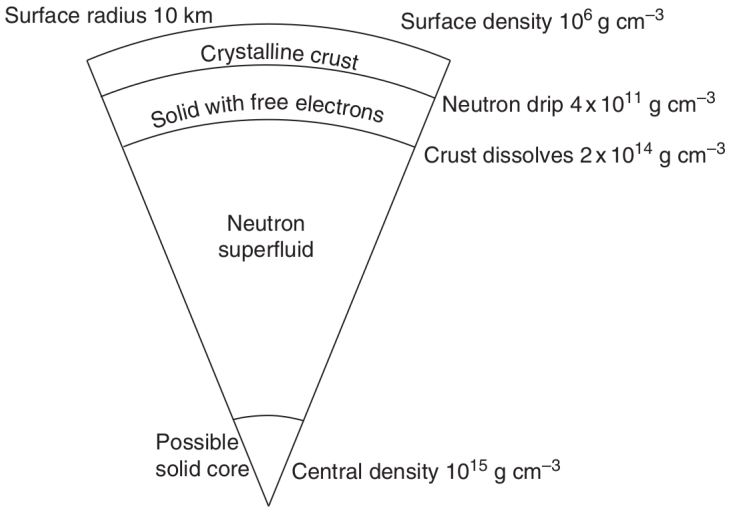

Figure 1.1: A simple presentation of the internal structure of a neutron star (Lyne & Graham-Smith, 2012).

1.2.4 Pulsar observations

1.2.5 Pulsars as a physics laboratory

The discovery of pulsars pioneered a new era of exciting sciences, addressing many challenges and unresolved questions in astronomy and fundamental physics. Studying pulsars helps researchers to understand the concepts behind various research topics, from planetary

physics to cosmology. MSPs (see Section 1.1.2) are particularly suited for the detection of gravitational waves due to their small spin period and stable rotation, which give very high precision in the measured pulse arrival times (Verbiest et al., 2009). In recent pulsar surveys, there has also been a strong drive to look for MSPs specifically to use them as probes to search for gravitational waves (Hobbs & Dai, 2017) since they have been shown to have exceptional spin period stability (Cordes & Shannon, 2010). This unique property of MSPs is critical for creating the so-called *celestial* clocks that are useful in pulsar timing research (e.g Hobbs et al. 2006). While there are currently more than a hundred MSPs being monitored regularly by pulsar timing arrays (PTAs, observations of a set of pulsars) program, there is still a need to uncover more suitable pulsars, as the sensitivity towards gravitational waves increases with an increasing number of pulsars being timed (Siemens et al., 2013). On the other hand, normal pulsars contributed to 90% of the pulsars population, therefore they are ideal in some case studies such as the validation of the model of pulsar evolution (Faucher-Giguère & Kaspi, 2006), and study of neutron star formation rate in the galaxy (Keane & Kramer, 2008).

1.3 Pulsars observed properties

The pulsed emission from pulsars is thought to be related to the loss of rotational energy due to their decreasing spin period. The pulsar magnetic field is considered to be a magnetic dipole M in a vacuum dipole model (Pacini, 1967), forming an angle α with the rotation axis, causing the star to lose energy through electromagnetic radiation due to its rotation. This appears to be the primary cause of the neutron star's energy loss. The variation of the angular velocity $\dot{\Omega}$ of the neutron star with respect to its moment of inertia I is defined in the vacuum dipole model as the following (Casini & Montemayor, 1998):

$$\dot{\Omega} = -\frac{2}{3}\frac{M^2\Omega^3}{Ic^3} \times \sin^2\alpha \qquad \text{or, simply} \qquad \dot{\Omega} = -\mu\Omega^{\mathrm{n}} \ . \tag{1.5}$$

In a more generalised case, the coefficients of Ω^3 in Equation 1.5 are defined by the parameter μ, which is also known as the net dipole moment of the neutron star (Ruderman et al., 1998). The parameter n is known as the braking index, where $n = 3$ is for the magnetic dipole model. The value of n can be derived from Equation (1.5), knowing the period and the spin-down rate. Different values of the braking index have been observed in isolated pulsars, which are not consistent with the value of n for the magnetic dipole model (e.g., Hamil et al. 2015). This deviation is detected for pulsars with frequencies ten times faster than slow rotating isolated pulsars.

1.3.1 Integrated profile and dispersion measure

The radio emission from the vast majority of pulsars is not bright enough to enable a significant detection of a pulse from just one rotation of the star. A technique called "folding" is therefore applied to increase the signal strength and to distinguish pulsar sources from background noise (Roberts et al., 2005). It consists of adding hundreds to thousands of pulses together to produce the so-called integrated pulse profile. The latter is known as the pulsar's fingerprint since each pulsar has its unique integrated profile shape. In comparison to individual single pulses, which are highly variable over time, the integrated profile is very stable. Pulsar searches exploit two key features of pulsars: their periodicity and the dispersion that occurs as signals travel through space. The arrival of the radio pulse from a pulsar is delayed due to free electrons from ionised gas in the interstellar medium. True pulsar signals are distinguished by these delays from terrestrial interference, which is frequently not dispersed. Pulse dispersion is a frequency-dependent delay in which a higher frequency signal travels faster than a lower frequency signal. Pulse dispersion is defined as the following (Lorimer, 2008):

$$\Delta t = 4.15\ ms \times \left[\left(\frac{\nu_{lo}}{GHz}\right)^2 - \left(\frac{\nu_{hi}}{GHz}\right)^2\right] \times \left(\frac{\mathrm{DM}}{\mathrm{cm^{-3}pc}}\right), \tag{1.6}$$

where ν_{lo} and ν_{hi} are respectively the low and high-frequency pulses, and DM is the so-called dispersion measure which can be used to infer the distance to the pulsar L, knowing the integrated column density of electrons, n_e (Shapiro & Teukolsky, 1983):

$$DM \equiv \int_0^L n_e\ dl \equiv \langle n_e \rangle L\ . \tag{1.7}$$

1.4 Formation and evolution

1.4.1 The P-Pdot diagram

The evolution history of pulsars is described by a diagram known as the $P - \dot{P}$ diagram (e.g., Johnston & Karastergiou 2017). Figure 1.2 shows that the diagram is built from the pulsar's spin period P as a function of the period derivative $\dot{P}$, also known as the spin-down rate, in logarithmic space. The diagram in Figure 1.2 clearly shows that pulsars fall into two main categories: normal pulsars (the dotted points in the diagram) and millisecond pulsars or MSPs (filling the bottom left of the diagram). Generally, normal pulsars have pulse periods $P \sim 0.5$ s with a period derivative $\dot{P} = 10^{-15}$ s/s, while MSPs rotate faster with a short period in the range $P \sim 1.4 - 30$ ms, and a period derivative of $\dot{P} \lesssim 10^{-19}$ s/s (Lorimer 2008). For example, the first observed normal pulsars and MSP respectively have periods $0.25 \lesssim P \lesssim 1.3$ s (Hewish et al. 1968; Pilkington et al. 1968) and $P = 1.558$ ms in case of PSR B1937+21 (Backer et al., 1982). In addition, lines of constant magnetic field strength and characteristic age are drawn in the diagram, which helps to visualise the effect of the variation of the spin period and the spin-down rate on the evolution of the pulsars. The

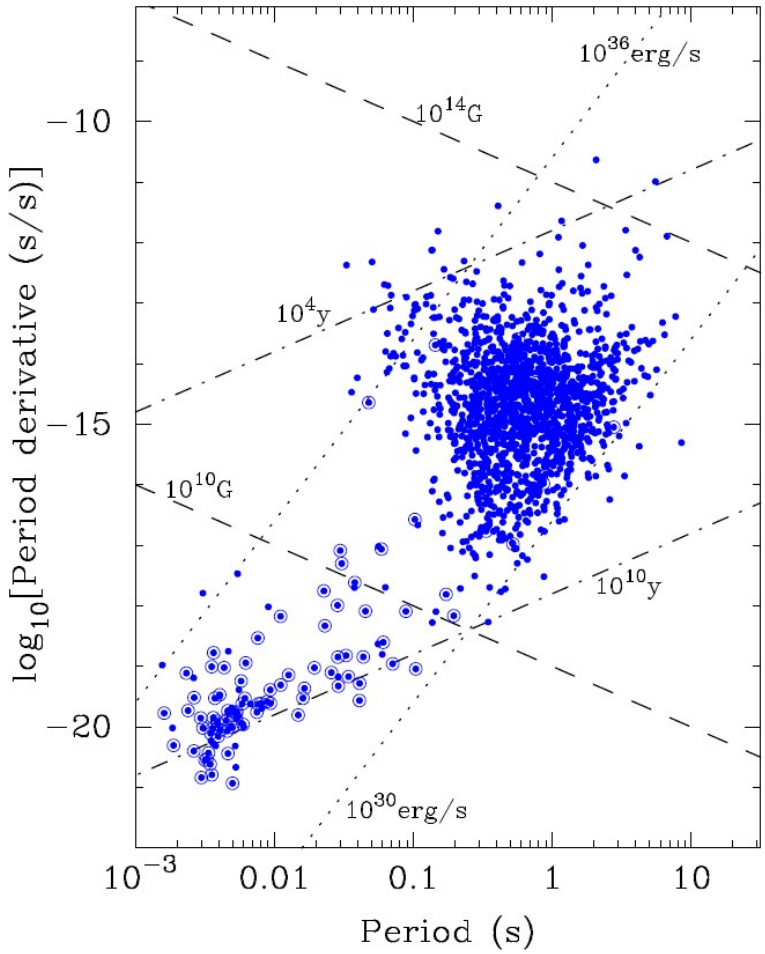

Figure 1.2: The $P - \dot{P}$ diagram (Lorimer, 2008), representing the population of pulsars, including normal pulsars and millisecond pulsars. Normal pulsars are marked by blue dots, while millisecond pulsars are indicated by circled dots. Lines of pulsars' constant magnetic field (dashed), characteristic age (dotted/dashed), and spin-down energy loss rate (dotted) are also drawn in the diagram.

magnetic field strength at the surface of the pulsars is derived using the relation (Kramer, 2004):

$$B_s = 3.2 \times 10^{19} \sqrt{P\dot{P}} (\text{Gauss}) \ . \tag{1.8}$$

On the other hand, assuming that the initial angular velocity of the pulsar is far larger than its present angular velocity, and taking into account the magnetic dipole model, the characteristic age of the pulsars can be estimated using the following relation:

$$\tau = \frac{P}{2\dot{P}} \ . \tag{1.9}$$

Furthermore, lines of a constant rate of energy loss $\dot{E}$, also known as the spin-down luminosity, are represented in the diagram. This energy loss is inversely proportional to the spin period P, and is given by: $\dot{E} \propto \dot{P}/P^3$ (Lorimer, 2008). This diagram reveals that the youngest normal pulsars are more energetic compared to MSPs.

1.4.2 Evolution of pulsars in a binary system

A neutron star is formed during the supernova explosion of a massive star in a binary system, as previously described in this section. According to the Virial principle, the binary system will be disrupted if more than half of the total mass of the supernova progenitor is expelled from the system during the explosion (Bhattacharya & van den Heuvel, 1991). The first evolution scenario involves a binary system that is unable to survive the explosion. In this case, an isolated neutron star and an OB star form as a result of the disruption. It is known that disruption events in binary systems are so frequent that only a few normal pulsars can be found in a binary (Lorimer, 2001). After $10 - 100$ Myr of the explosion, the pulsar would become a slow-rotating radio pulsar, and eventually, its energy would not be sufficient to normally produce radio emission (Lorimer, 2008). The second scenario occurs when the binary system survives the explosion: the neutron star accretes matter from its evolving companion. On one hand, the companion explodes if it consists of a high-mass system. Again, if the system survives the explosion, it forms a binary system of two neutron stars, also called *binary neutron star*. Otherwise, the system is disrupted into young pulsars and the so-called *recycled pulsars*. The latter is triggered by the old neutron star which was revived by obtaining angular momentum from the system. On the other hand, if the companion is a low mass object, it evolves into a white dwarf and the system forms a *millisecond pulsar - white dwarf binary*. In the high mass system, the accretion duration is short before the star explodes, and the pulsar is mildly "spun up" and has a period of a few 10's of milliseconds. If the companion is smaller and does not explode, the accretion can last longer and the period is instead a few milliseconds.

1.4.3 Pulsar search

Pulsar searches are continuing to find more extreme systems to test theories of gravity, better systems for detecting gravitational waves, and to understand more about the types and evolution of pulsars and neutron stars. Over the last few decades, a large number of pulsar surveys have been conducted, allowing us to gain a better understanding of the pulsar population. The Square Kilometre Array (SKA) project will be an incredible tool for pulsar searching. This project aims to construct the largest radio telescope in the world, ultimately with a collecting area of over a square kilometre[1]. The SKA's extraordinary sensitivity and wide field of view will be vital for pulsar science and many related fields. The sensitivity of the first phase of the telescope (SKA1) will find enough pulsars to quadruple the known pulsar populations, including pulsars in other galaxies (Levin et al., 2017). When the whole SKA is fully operational, it will be sensitive enough to find nearly all of the known types of pulsars in our galaxy, including millisecond pulsars, rotating radio transients, and pulsars in binary systems. By observing pulsars with SKA, the precision of measuring the pulse arrival time of the brightest pulsars will be increased by a factor of 100, making pulsars the

[1] `https://www.skatelescope.org/the-ska-project/`

best instrument for absolute time calibrations (Rea & Pons, 2015). However, the daily 60 PetaByte (PB) of pulsar search data requires adequate tools for pulsar identification and real-time processing (Levin et al., 2017). Pulsar search pipelines and pulsar detection tools need to be developed to address this huge amount of pulsar data.

1.4.4 The High Time Resolution Universe survey

We present the High Time Resolution Universe (HTRU) survey, in particular HTRU-S, because of the large volume of candidates it produced, but also because there are a number of studies that have used it to test and improve Machine Learning classification codes. This has been enabled by the data sets being made available publicly (e.g., HTRU1 by Morello et al. (2014) and HTRU2 by Lyon et al. (2016)) or shared with us by Balakrishnan et al. (2021). This public sharing of data has resulted in tens of publications and has greatly benefited this work as well.

The HTRU survey was an all-sky survey conducted from November 2008 to January 2014. This survey made use of two systems: the 64-m Parkes Multi-Beam system, which was used to search the Southern galactic plane region, referred to as the HTRU-S survey (Keith et al., 2010); while the 100-m Effelsberg 7-beam system is used to observe the Northern galactic plane, referred to as the HTRU-N survey (Barr et al., 2013). The survey sensitivity is supposed to be similar for both galactic surveys. The frequency and time resolution of this survey is much larger than its predecessor, the Parkes Multi-Beam Survey. As a result, they were less susceptible to dispersive smearing and will be able to detect a greater number of millisecond pulsars. Hundreds of millisecond pulsars would more likely be discovered in these surveys. The two key goals of the HTRU survey were to extend the understanding of the MSP population and to characterise the transient sky on timescales as short as tens of microseconds (Keith et al., 2010).

1.5 Pulsar detection tools

The number of detected candidates in pulsar surveys has increased dramatically over time. The number of discovered pulsars, on the other hand, is not proportional to the number of candidates discovered during the survey. For instance, the early generation of pulsar surveys produced only a few thousand candidates, resulting in the detection of a very high proportion, 224, pulsars (Manchester et al., 1978). While the next generation of surveys produced millions of candidates, such as the Parkes Multibeam Survey (PMPS, Manchester et al. 2001), which produced approximately 8 million candidates, including 833 pulsars; and the HTRU survey, which detected 495 pulsars among the 55 million candidates (Lyon et al., 2016). This increase in candidate numbers is due to having to go to ever greater sensitivity to find new pulsars, and longer integration times, wider bandwidths, and so many more trials are being performed in pulse period and dispersion measure space. As a result, pulsar candidate

selection methods are becoming more widely recognised as a major source of concern in pulsar search. While manually inspecting each candidate based on the pulse profile and S/N was once possible (Clifton & Lyne, 1986), it is now impractical due to a large number of candidates to be investigated. Some techniques have been developed to reduce the number of candidates to be inspected, including graphical tools (e.g., REAPER, Faulkner et al. 2004) and semi-automated pulsar candidate ranking (PEACE, Lee et al. 2013). Graphical tools can effectively boost the detection of real pulsars (Faulkner et al., 2004). However, they are subject to errors and have limitations (Bates et al., 2012). To further improve the detection of real pulsars, automated pulsar candidate selection has gained popularity recently. It involves using the so-called machine learning technique (described in Chapter 2) to automatically classify real pulsars from other sources of noise.

1.6 Problem statement

As discussed in the previous sections, modern large-scale pulsar surveys generate large numbers of candidates due to the increase in telescope sensitivity and increased search parameter space. This results in an enormous volume of unlabelled datasets to be processed and classified by astronomers, coining the term "candidate selection problem" (Lyon et al., 2016). Since it is mostly impossible to visualise and classify millions of pulsar candidates from pulsar surveys, machine learning classifiers are intended to reduce the number of candidates to be inspected by human experts. This approach has been used in the pulsar sifting of 17.6 million candidates, which has led to the discovery of 23 new pulsars (Morello et al., 2014) in the HTRU-S survey. Although this approach is effective, some challenges are frequently encountered in the classification of pulsars using machine learning. First, in pulsar datasets, noise and radio frequency interference (RFI) account for the majority of candidates, resulting in a machine learning problem known as *imbalanced learning problem* (He & Ma, 2013). The latter reduces the classification performance of a machine learning classifier, which means that more pulsars are missed by the classifier or many non-pulsars candidates are classified as real pulsars. Besides, in the view of future pulsar surveys using the SKA telescope, there is a strong need for real-time machine learning in addition to the imbalanced data issue. Second, there are still only a few labelled pulsar candidates available in the field of pulsar astronomy, which limits the performance of a machine learning classifier. One of the reasons is that pulsars are hard to detect due to their faint signals, and as a result, specialised machine learning classification systems have been developed for this domain, and this remains an active research area (e.g., Lyon et al. 2016; Tan et al. 2018). Despite the success of these studies, reducing the number of false/noise candidates while increasing the number of detected true pulsars remains ongoing research. Successfully resolving the class imbalance problem has the potential to significantly improve the accuracy of modern machine learning systems applied to pulsar classification, which could lead to systems that make ground-breaking discoveries in pulsar astronomy.

1.7 Scope of the project

Many efforts have been made to tackle the imbalanced class problem in the field of machine learning. Some of them involve balancing the data using different approaches, including data re-sampling (Ando & Huang 2017; Buda et al. 2017), synthetic sample generation (Kingma & Welling 2013; Goodfellow et al. 2014), and cost-sensitive re-weighting (Byrd & Lipton, 2019). In addition, the so-called Generative Adversarial Network (GAN, Goodfellow et al. 2014) is one of the synthetic data generation approaches that has piqued the interest of computer vision researchers. It can produce high-resolution synthetic images and has helped to overcome the imbalanced learning problem in various research domains. Several of these studies have been applied in pulsar searches (e.g., Lyon et al. 2016; Tan et al. 2018; Lin et al. 2020b). However, the presence of an imbalanced class in pulsar datasets is a remaining constraint to the detection of new pulsars. This thesis attempts to alleviate the problem of class imbalance in pulsar candidate classification and also investigate machine learning techniques that address the limitation of labelled pulsars. To achieve this goal, we introduce various machine learning techniques that could potentially be used for pulsar candidate classification in Chapter 2. In Chapter 3, we implement some of these techniques for solving pulsar candidate classification in highly imbalanced class datasets. In Chapter 4, we explore machine learning techniques that address the lack of labelled pulsar candidates. The summary of our findings, as well as future work, are presented in the conclusion section.

Chapter 2

Artificial intelligence, Machine Learning, and deep learning concepts

2.1 Overview

Artificial intelligence (AI) has advanced rapidly in various fields of research, including computer science and technology, over the last decade. Robotics (Joshi et al., 2015), speech recognition (Hawley et al., 2013), object recognition (López-Antequera et al., 2019), and language processing (Devlin et al., 2018) are among AI's biggest achievements. There are many ways to define AI, for instance, Sun et al. (2019) defines AI as "the process of simulating human consciousness and thinking, mainly used to research and develop technologies, theories, and methods related to human behavior and thinking". The first approach to mimic human intelligence is known as symbolic AI (Newell, 1982). This approach consists of creating a set of rules or codes so that the computer can solve a well-defined problem, such as finding a root of an equation. However, the symbolic AI approach is limited to solving logical problems. The so-called machine learning (ML), which can tackle more complex problems like image classification, is an approach that has taken AI to the next level (Mitchell, 1997). A machine-learning system, unlike symbolic AI or classical programming, learns a set of rules through experience by mapping input data to the expected output for a specific task (see Figure 2.1). The learned rules are in turn applied to unseen data to yield an output for this specific task. The term *training an algorithm* or *a model* refers to this process of learning. Specifically, a machine learning algorithm automatically finds the best representation of the input data by transforming or encoding the data appropriately to achieve a specific task such as a classification or a regression problem. In some cases, one representation of data is not sufficient to best describe the data space. This implies that a more representative data space is needed to capture the entire structure/pattern of the data, and this can be achieved by layers of representations, also known as *deep learning* (DL). Deep learning is considered to be a subfield of machine learning. Deep Learning transforms the data through some successive layers of representations stacked together in a model referred to as *Artificial*

Neural Networks (ANNs). More details on this will be discussed in Section 2.3.1. Machine learning tasks can be classified into three main categories: supervised, semi-supervised, and unsupervised learning algorithms.

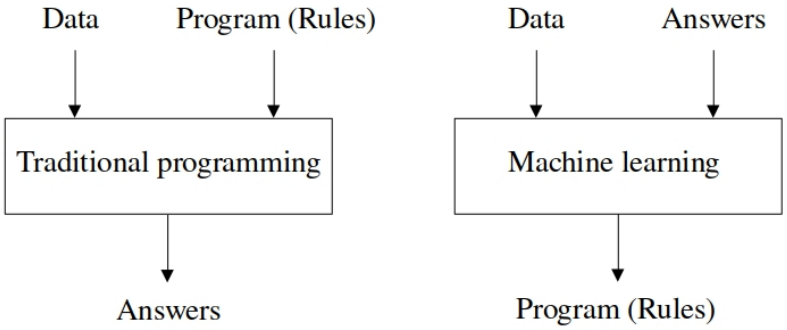

Figure 2.1: A comparison between classical programming and a machine learning approach. The first one aims to solve a specific problem through input data and a set of rules, while the machine learning approach learns from the input data and the expected outputs to produce a set of rules or models that can be used to predict the outputs of unseen input data.

2.1.1 Supervised learning

Supervised machine learning has been proven to be effective in a variety of computational tasks from object detection (e.g Zhao et al. 2017; Han et al. 2015) to speech recognition (e.g Wang 2020; Graves & Jaitly 2014). Training examples are mathematically represented by a combination of two vectors (x, y) where x is a multi-dimensional vector ($x \in \mathbb{R}^n$), known as features or input vectors, and y is referred to as target or output vectors. Generally, a dataset can be described as an ensemble of a pair of vectors $V = \{(x_1, y_1), ..., (x_N, y_N)\}$, where N is the size of the overall examples in the dataset. In supervised learning, an algorithm is trained on labelled data (x, y) or training examples to build a model that can be used to classify or to make a prediction of unseen data. A machine learning algorithm aims to learn a mapping function $y = f(x)$, from the input data to the output vectors, and to find the best approximation of this function to obtain a model that can predict the expected output values of new input data. A supervised learning task can be either a classification problem, where the model predicts the output value (discrete values) of the input vectors, or a regression problem, where, for given inputs, the model attempts to predict the output vectors (continuous values).

2.1.2 Semi-supervised learning

Semi-supervised learning (SSL) is a machine learning framework designed to solve machine learning tasks with a limited number of labelled examples together with a large number of unlabeled examples (Chapelle et al., 2010). Data labelling is often expensive, time-consuming,

or requires human effort in real-world applications. For instance, labelling hundreds of thousands of pulsar candidates is time-consuming in pulsar astronomy, as it requires human expert validation to determine whether the candidate is genuine (Morello et al., 2014). SSL provides the algorithm with a small number of labelled examples from which to learn, while classifying the unlabelled examples as accurately as possible. As a result, SSL compensates for the lack of labelled data and has been shown to perform well with only a few labelled examples (e.g., Odena 2016; Salimans et al. 2016). SSL comes in a variety of algorithms, including Transductive Support Vector Machines (Joachims 1999; Chapelle et al. 2010) and label propagation algorithms (Zhu & Ghahramani, 2002). The utility of SSL in combination with a generative models approach to solve a classification problem is explored in Chapter 4.

2.1.3 Unsupervised learning

Unsupervised learning is another class of machine learning algorithms that aims to learn robust representations of data (features) independently of human annotations. In other words, no labelled examples are provided during the training process, and the algorithm finds any possible structure or patterns in the data. The learned features can be used for various applications, such as *clustering*, where the algorithm finds groups of similar examples in the data; and transfer learning (He et al., 2020), where pre-trained features are used to solve downstream tasks (e.g., object detection; image segmentation). Unsupervised learning is largely used in different machine learning applications, such as object detection (Sermanet et al. 2012; Zhao et al. 2018) and natural language processing (Radford & Narasimhan 2018; Devlin et al. 2019).

2.1.4 Regression

A regression algorithm is applied when the output values are continuous. For instance, a regression model can be used to predict the future price of a stock or to forecast temperature over a period of time. Regression models include linear regression, LASSO regression, and ridge regression, among others (Forsyth, 2019). Linear regression is the most widely used regression algorithm where the input and output values are fit with a linear model. The goal of linear regression is to minimise a loss function to reduce the mean squared error (MSE) between predicted and actual output values. The model and the loss function for this algorithm are described as the following:

- Model: $\hat{y} = \sum_{i=1}^{n} \theta_i x_i + b$
- Loss function: $MSE = \frac{1}{n} \left(y - \sum_{i=1}^{n} \theta_i x_i + b\right)^2$,

where n represents the number of the examples, x_i is the input of the ith training examples, and θ and b are respectively known as feature weights and biases. The value of θ that

minimises the loss function can be directly determined using the *normal equation*:

$$\hat{\theta} = (X^T X)^{-1} X^T y \ . \tag{2.1}$$

However, the computational complexity of inverting the X matrix, which is time-consuming for a large number of features, is a major disadvantage of this approach. This problem can be avoided by minimising the loss function using a technique known as gradient descent optimization (See Section 2.3.2).

2.2 Classification

Classification is a predictive modelling problem in which a model is trained to learn a class label for a given input example. In other words, the algorithm is trained on the training examples to learn a mapping relationship between input data and class labels. There are many types of classification tasks. In this section, we discuss mainly about binary and multi-class classification. The first corresponds to a scenario where the data contains only two classes of examples, whereas the latter consists of more than two classes. Examples of multi-class classification tasks in Astronomy include a classification of galaxy morphology into a spiral, elliptical, and barred spiral (Barchi et al., 2020), and a classification of astronomical objects into stars, galaxies, and quasars (Clarke et al., 2020). On the other hand, an example of binary classification task is the classification of gamma-ray sources into AGNs and pulsars (Bhat & Malyshev, 2021). Different types of ML algorithms (also known as classifiers) can be used for classification tasks, which are discussed in the subsequent sections. From a statistical perspective, these classifiers fall into two categories (Braga-Neto & Dougherty, 2020): a probabilistic machine learning model, such as Naive Bayes classifiers and Bayesian neural networks, and a deterministic model (e.g., a standard Support Vector Machine or SVM algorithm). It could be possible, however, to calibrate a deterministic model to have a probabilistic output and vice versa. For instance, a standard SVM algorithm can be converted to output a class probability Nalbantov & Ivanov (2019). Considering a multi-classification task with N classes, the probabilistic model outputs the probability that a given input belongs to each of the N classes. In the previous example, if the input is an image of an astronomical object, the model produces three probabilities: stars (0.7), galaxies (0.2), and quasars (0.1). The classifier is more confident, in this example, that the input image is most likely in the Stars class. The main advantage of using probabilistic models is that the uncertainty associated with the prediction is known. Therefore, probabilistic models are crucial in ML applications such as medical diagnosis (Vega et al., 2021) and autonomous driving (Melotti et al., 2020). Unlike the probabilistic model, the deterministic model predicts the most likely class of the input data (e.g., Stars).

In this thesis, we investigate different machine learning algorithms for pulsar candidate classification using both supervised and semi-supervised approaches. In Chapter 3, we apply

various supervised learning techniques to pulsar classification problems, such as ANNs and Random Forest. These techniques are described in the following sections. However, most of these approaches are only limited to feature extracted from the data. Therefore, we explore ML techniques that can be used for training image datasets, which are described in Section 2.3. In addition, since the annotation phase of pulsar candidates is highly time-consuming, in Chapter 4 we investigate a semi-supervised learning approach using generative adversarial networks (See section 2.4).

2.2.1 Support vector machine

Support Vector Machine is a kernel-based learning algorithm for classification and regression (Vapnik, 2000). SVM has gained popularity as an ML algorithm for classification (e.g., Dorn-Wallenstein et al. 2021; Gayatri et al. 2021) due to some factors: its robust generalisation ability, its good performance in high-dimensional spaces, its memory efficiency, and its small number of learnable parameters (Cervantes et al., 2020). SVM maps the input data to a new high-dimensional space, known as feature space. The ultimate goal of SVM is to find a good decision boundary that can discriminate the data points (see Figure 2.2). A decision boundary is represented as a hyperplane (or a line for two-dimensional space) that separates different classes of data points in the feature space. This hyperplane is an N-dimensional space in the case of a dataset with N features. There can be an infinite number of hyperplanes that could separate every class of data points, but the ideal decision boundary corresponds to the hyperplane with maximum margin, known as optimal separation hyperplane (Cristianini & Shawe-Taylor, 2000). To obtain the optimal separation hyperplane, the algorithm seeks to maximise the distance between the hyperplane and the closest data points to the hyperplane, called *support vectors*. The classification is done by projecting new data points to either side of the hyperplane, and their class is assigned based on their position relative to this hyperplane. In practice, the data points are not always linearly separable, as shown in Figure 2.3. It is therefore impossible to separate the data points with a straight line. In this case, SVM uses a technique called *kernel trick* which consists of transforming the initial feature space (or input space) into a new higher-dimensional feature space, where the optimal decision boundary can be found (e.g., Figure 2.3). The transformation is performed by invoking the so-called *kernel function*, which maps the input data points to their new representation space. Depending on the case study, there are numerous kernel functions to choose from, including the following (Patle & Chouhan, 2013):

- Polynomial kernel:

$$K(x, y) = (\alpha x^T y + c)^d \ ,$$

- Radial Basis Function (RBF) kernel:

$$K(x, y) = e^{-\alpha \|x-y\|^2}, \ \alpha > 0 \ \ ,$$

- Sigmoid kernel:

$$K(x, y) = \tanh(\alpha x^T y + c) \quad ,$$

where d is the degree of polynomial, c a constant term, and α adjustable parameters of the SVM model. While SVM has a high degree of generalisation and training efficiency, it is frequently hard to find the optimal kernel function and model hyper-parameters for a particular problem. Furthermore, for a large training dataset, the training time is very long (Gholami & Fakhari, 2017).

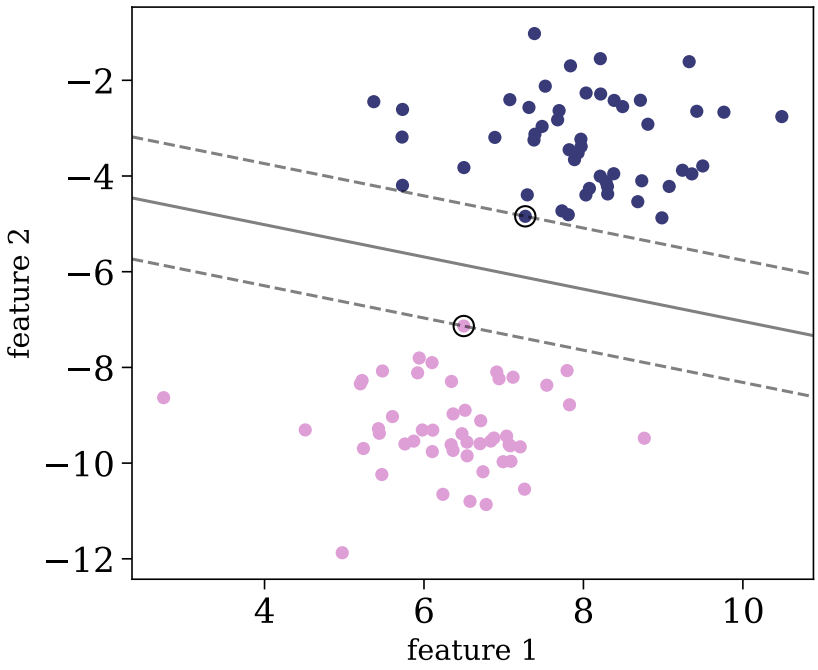

Figure 2.2: An example of two-dimensional feature space for SVM classification (Baron, 2019): the pink and purple points correspond to two different classes. The straight solid line represents the optimal decision boundary. The two dashed lines are known as the negative and positive margins, and the two encircled points are defined as the support vectors.

2.2.2 Ensemble learning algorithms

Ensemble learning (Maclin & Opitz, 2011) is a well-known machine learning technique that consists of merging multiple weak classifiers (learners), also known as base learners, to produce a more accurate classifier. The primary advantage of the ensemble learning method is that not only does it mitigate the overfitting problem (see Section 4.4.1), but also it generalises well on new unseen data (Zhou, 2012). Besides, ensemble learning classifiers address the main challenges of training a machine learning algorithm: bias and variance (Geman et al., 1992). The term “bias” refers to the difference between the model’s average prediction and the actual target value. A model with a significant bias is frequently associated with a model

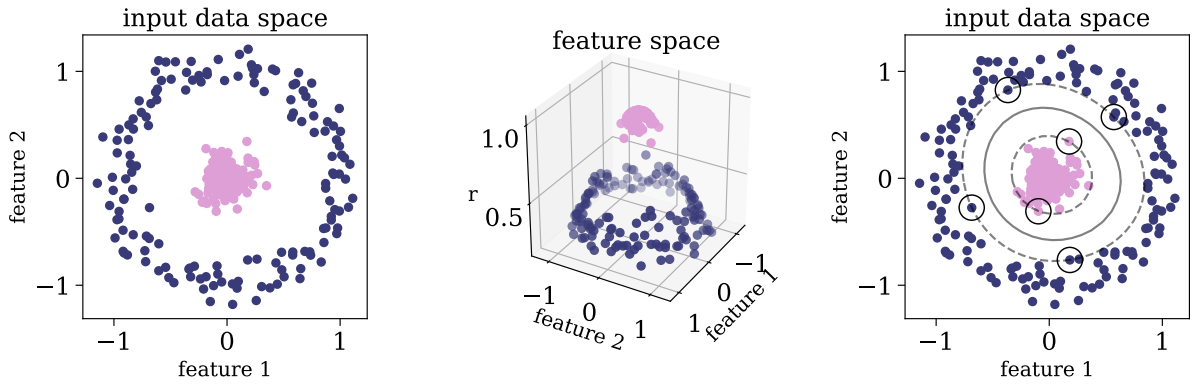

Figure 2.3: An example of an SVM kernel trick (Baron, 2019): the left panel shows a two-dimensional feature space or input data space where two classes of data points (pink and purple) cannot be linearly separable. The middle panel illustrates the kernel trick, which transforms the initial feature space to a three-dimensional space, where a linear separation is possible by using a two-dimensional hyperplane. The right panel represents the decision boundary and support vectors derived from applying the kernel trick in the new feature space.

with a low generalisation capacity. In other words, the model is unable to learn sufficiently from the training data. The variance problem, on the other hand, refers to the sensitivity of a machine learning algorithm to a slight variation of the training examples. A sensitive model generates different results each time it predicts the target of the same instance. Variance can be defined as a measure of the variation of the model's predictions on a given instance. The challenge is that minimising variance often results in highly biased models, while reducing bias may result in a high variance in the model predictions: this is referred to as the *bias-variance trade-off*. The goal of any machine learning algorithm is to produce a model with a low bias and a low variance. There are several types of ensemble learning methods, including bagging (Breiman, 1996a), boosting (Schapire 1990; Freund & Schapire 1996), and stacking (Wolpert, 1992). These approaches differ as to how the results from the weak classifiers are aggregated. Bagging and boosting are considered to be effective methods for addressing the bias and variance problems (Breiman, 1996b).

2.2.3 Bagging and boosting ensemble learning

Bagging is a form of ensemble learning method in which many classifiers are trained simultaneously on a randomly selected collection of training samples. This approach uses the so-called bootstrapping technique (Efron & Tibshirani, 1986) when selecting training examples for each classifier. A bootstrap sample is obtained by resampling the original training set with replacement; some instances may be repeated in the same training set, while others may be left out. As a result, each of the weak classifiers is trained on a unique set of random examples. The latter frequently results in a high variance weak classifier, where a slight change in the training set might affect the classification results adversely or favourably. Training these weak classifiers with the bagging ensemble produces a single classifier with low

variance. It is suggested that bagging is an effective method for unstable learning algorithms such as neural networks and decision trees (DT, Breiman 1996c). Bagging aggregates the results from each weak classifier and makes a final prediction based on a *voting* method; a class label that the majority of the base learner predicts is assigned to a given instance. On the other hand, the boosting ensemble method trains classifiers sequentially, where each subsequent classifier attempts to correct the prediction errors from the previous classifier. In other words, the next classifier focuses on instances that the previous classifier failed to classify. Unlike the bagging approach, which may include repeated examples in the training set, the boosting method carefully selects the training set, ensuring that examples misclassified by previous classifiers are either included more often in the next training set or given more weights. The latter results in highly biased, low variance weak classifiers. As with bagging, boosting ensemble combined these ineffective classifiers to produce a more accurate single model with low bias.

Implementing such ensemble methods results in a more stable final model than merely using a single model (Zhou, 2012). Ensemble learning has been used to solve various challenges in Astronomy, such as pulsar classification (e.g., Lyon et al. 2016; Tan et al. 2018; Lin et al. 2020a), stars-galaxy classification (Kim et al., 2015), and outlier detection (Nun et al., 2016). A widely-used base learner in ensemble learning is the DT algorithm (Quinlan, 1986).

2.2.4 Decision tree algorithm

Decision trees are tree-based algorithms used for classification and regression tasks (Quinlan, 1986). A DT is mainly built with a root node, decision nodes, and leaves, as illustrated in Figure (2.4). The goal of the DT algorithm is to learn decision rules from the training examples and to be able to predict the class of new examples based on the learned rule at each decision node. These decision rules are represented as a threshold which is set at each decision node or feature variable. For a binary classification problem, the training process of the tree algorithm starts by searching for the most discriminative feature variable X_i with a threshold $X_{i,thres}$. This feature is used as a splitting point at the root node in which two new decision nodes or child nodes are created. The training examples are propagated to these child nodes based on the threshold value. Two more nodes are produced from each of these child nodes, and the process repeats until the nodes are pure: contain only a single class label. These nodes are known as the leaf nodes where the class label of each example is assigned. It is a difficult task to determine the optimal features for class separation within each node. Therefore, statistical tools are used to determine the relative importance of each feature, including information gain (IG, Mitchell 1997) and Gini impurity (Breiman et al., 1984). The feature with the highest importance score is located at the top of the tree, while those with lower importance scores are located in the tree's lower branches. For a given feature variable X_i, the IG measures the reduction in entropy of a subset of examples S_ν

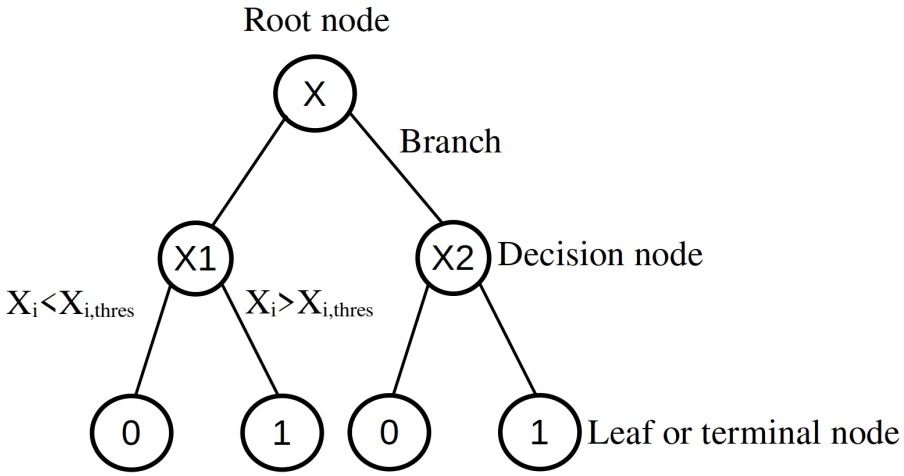

Figure 2.4: An illustration of a simple decision tree structure for a binary classification. The tree is characterised by a root node at the top of the tree, followed by a branch/sub-tree, a decision node, and a leaf/terminal node at the bottom of the tree. The root node contains the entire sample and is split into two or more branches. Each decision node represents a feature variable X_i where a decision split-point is made based on the value of $X_{i,thres}$. The leaf contains the final class label (0,1).

drawn after splitting a set of examples S:

$$IG(S, X_i) = Entropy(S) - \sum_{\nu \in Values(X_i)} \frac{|S_\nu|}{|S|} \, Entropy(S_\nu) \; . \tag{2.2}$$

Where $Values(X_i)$ is the set of possible values in the feature variable X_i. The first term in Equation 2.2 denotes the entropy before splitting the initial set of examples, while the second term denotes the weighted entropy after splitting. It is also known as the Shannon entropy in information theory, and it expresses the amount of uncertainty/information that can be derived from a probability distribution for a specific event. In the context of machine learning, entropy is defined as the measure of impurity or homogeneity in a random set of examples:

$$Entropy(S) = \sum_{i=1}^{k} -p_i \; log_2 \; p_i \; . \tag{2.3}$$

Where the parameter k corresponds to the number of class labels in the training examples, and p_i the proportion of examples in class i. Any features with the highest IG are considered to be the most discriminative features. IG is used in popular DT algorithms such as Iterative Dichotomiser 3 (ID3, Quinlan 1986) and C4.5, (Quinlan, 1993). The Gini impurity, on the other hand, is used in the so-called Classification and Regression Tree (CART) algorithm (Breiman et al., 1984). The Gini impurity quantifies the likelihood that a set of examples D would be incorrectly labelled if it were randomly labelled according to the distribution of the

classes in the dataset. (Bonaccorso, 2017):

$$Gini\ Impurity(D) = 1 - \sum_{i=1}^{k} p_i^2 \ . \tag{2.4}$$

Where p_i is the proportion of the k^{th} class of examples in the dataset D. If a subset of data has a Gini impurity value of 0, it is considered pure. The main advantage of the DT algorithm is that it uses a small number of hyperparameters, which enables it to be easily trained on large datasets. Additionally, because this algorithm generates feature importance scores, it can be used as a feature selection tool (e.g. Bethapudi & Desai 2018).

2.2.5 Random Forest

A single decision tree is frequently overfitting when the number of nodes or the depth of the tree structure increases. Training a collection of decision trees, known as Random Forest (RF), can alleviate the problem of overfitting (Breiman, 2001). RF is a bagging ensemble learning method composed of a collection of decision tree algorithms. Each tree is trained independently using the bootstrapping technique on randomly selected subsets of training examples and features. The prediction of previously unseen data is based on a majority vote of each tree prediction in the forest as presented in Figure 2.5, and given by the following relationship for a binary classification task (Wang et al., 2019a)):

$$\hat{y} = \begin{cases} 1, & \sum_{i=0}^{k} y_i >= \theta \\ 0, & \sum_{i=0}^{k} y_i < \theta \end{cases} \ , \tag{2.5}$$

where $\hat{y}$ is the final prediction and y_i is a prediction of i^{th} tree. The class label is determined by a voting threshold θ. RF offers some hyper-parameters that can be optimised during training, such as the number of features and trees. The resulting performance of the final classifier on new data is usually better than a single decision tree classifier (Breiman, 2001).

2.2.6 XGBoost

eXtreme Gradient Boost (XGBoost, Chen & Guestrin 2016) is a tree-based algorithm that belongs to the class of boosting ensemble learning (Schapire, 1990). The idea of XGBoost was inspired by previous boosting techniques such as Adaptive Boosting (Adaboost, Freund & Schapire 1997) and Gradient Boosting Classifier (GBC, Friedman 2001). XGBoost works similarly to these approaches while offering some improvements to the former methods: first, XGBoost adds regularisation techniques, known as L1 and L2 regularisation, to reduce the overfitting problem. Second, XGBoost can boost the training time by parallelising tree processing tasks and by making a Graphical Processing Units (GPUs) feature available during the training. XGBoost prediction is a weighted-sum of all weak learners (Chen & Guestrin,

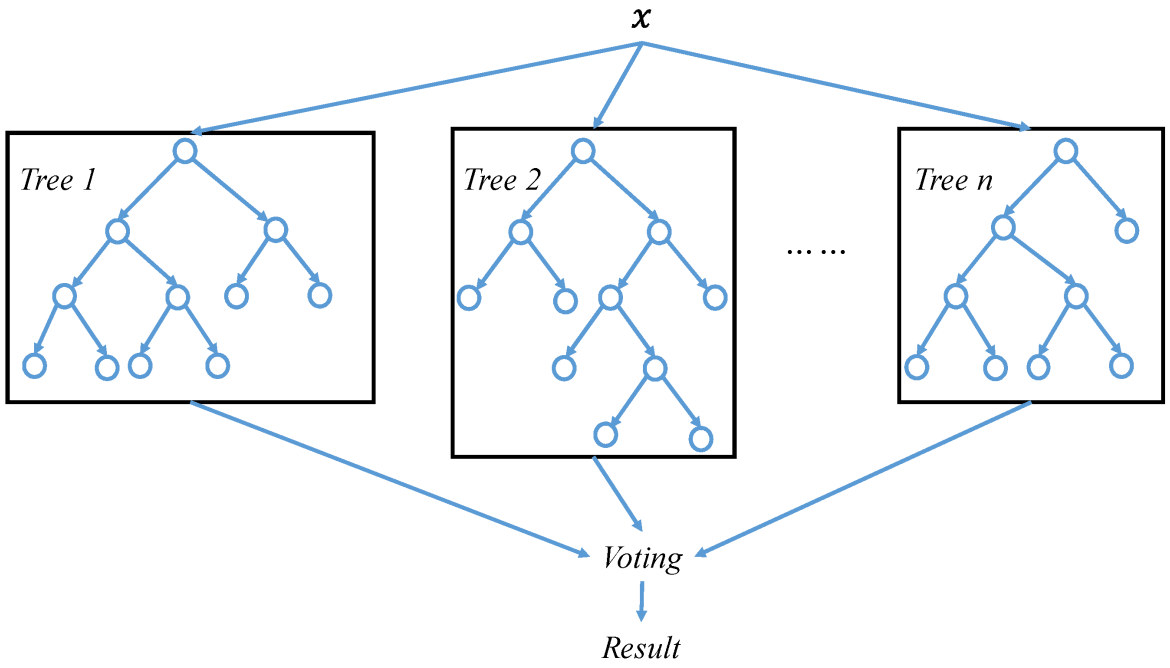

Figure 2.5: Structure of a Random Forest classifier with n number of trees (Wang et al., 2019a). Each tree predicts the class label of an instance x, and the final class prediction is made by a majority vote.

2016):

$$\hat{y}_i = \sum_{k=1}^{n} f_k(x_i), \tag{2.6}$$

where n is the number of trees in the ensemble learner, f_k represents a tree, x_i is the i^{th} training example. Thus, $f_k(x_i)$ corresponds to the prediction value from the k^{th} tree as illustrated in Figure 2.6. Both RF and XGBoost are powerful ensemble learning tools that are widely used in diverse areas of research including Astronomy (e.g., Lin et al. 2020a; Wang et al. 2019a; Golob et al. 2021). Furthermore, these ensemble learning methods are easy to implement and effective on large datasets because of their generalisation performance Wang et al. (2019a). However, RF and XGBoost, along with other algorithms such as SVMs, are not effective at handling perceptual machine learning problems such as image classification and speech recognition (Chollet, 2017). Therefore, it is necessary to explore the concept of deep learning which allows such problems to be solved effectively. The next section focuses on neural network models and their application to image classification.

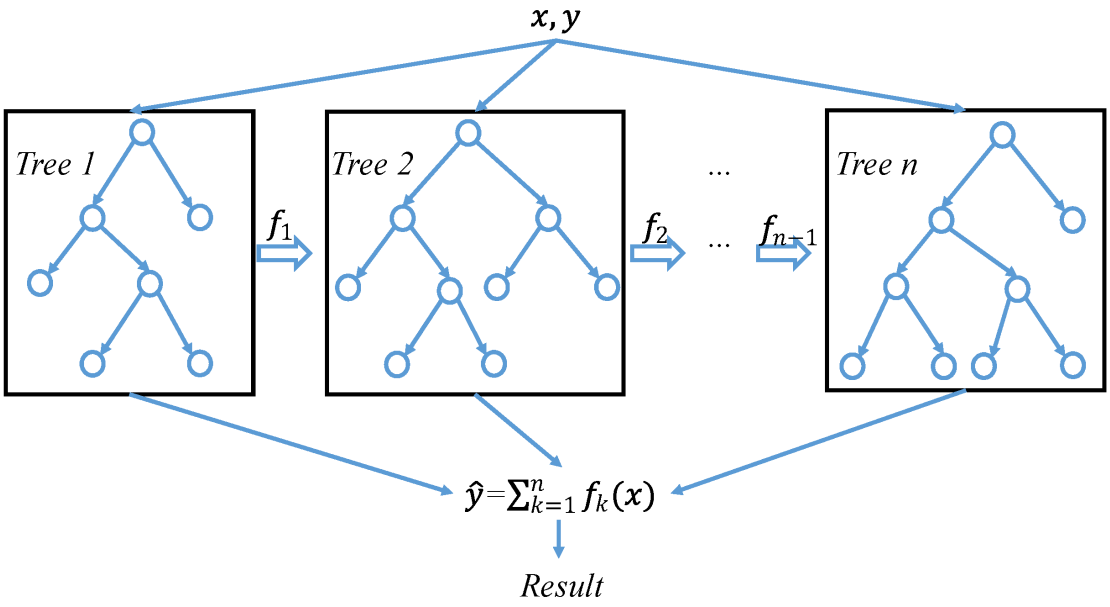

Figure 2.6: Structure of an XGBoost classifier with n trees (Wang et al., 2019a). Each tree is trained sequentially on a training set (x, y), with each subsequent tree focusing on the error associated with the previous tree's prediction f_k. The final classification result is the weighted sum of the tree predictions.

2.3 Neural networks

2.3.1 Artificial Neural Network

As previously introduced in section 2.1, an Artificial Neural Network (ANN), or simply a neural network, is a deep learning model that is trained through successive layers of data representations. ANN simulates the general function of the human brain by mimicking how neurons in the brain interact. The network is made up of three main components: an input layer, an output layer, and a series of hidden layers in between. A multi-layer perceptron (MLP) is a network that contains multiple hidden layers. Each layer consists of multiple interconnected neurons, which are the building blocks of the neural network. There are many different architectures of ANNs, but the most basic is known as the feed-forward neural network (Schmidhuber, 2015). In this network, input variables are forwarded from the input layers to the output layers through a series of transformations or activations at each neuron (Bishop, 2006):

$$a_j = \sum_{i=1}^{n} w_{ij} x_i + w_{j0} \ , \tag{2.7}$$

where a_j is the activation of a neuron j, x_i the input variables from the previous n neurons, and w_{ij} and w_{j0} are adjustable parameters of the model, known as weights and biases respectively. As shown in Figure 2.7, these activations a_j are then transformed by a non-linear

function h, referred to as an activation function, to provide outputs z_j to the next layers:

$$z_j = h(a_j). \tag{2.8}$$

This function is an essential part of the network since a non-linear transformation of the

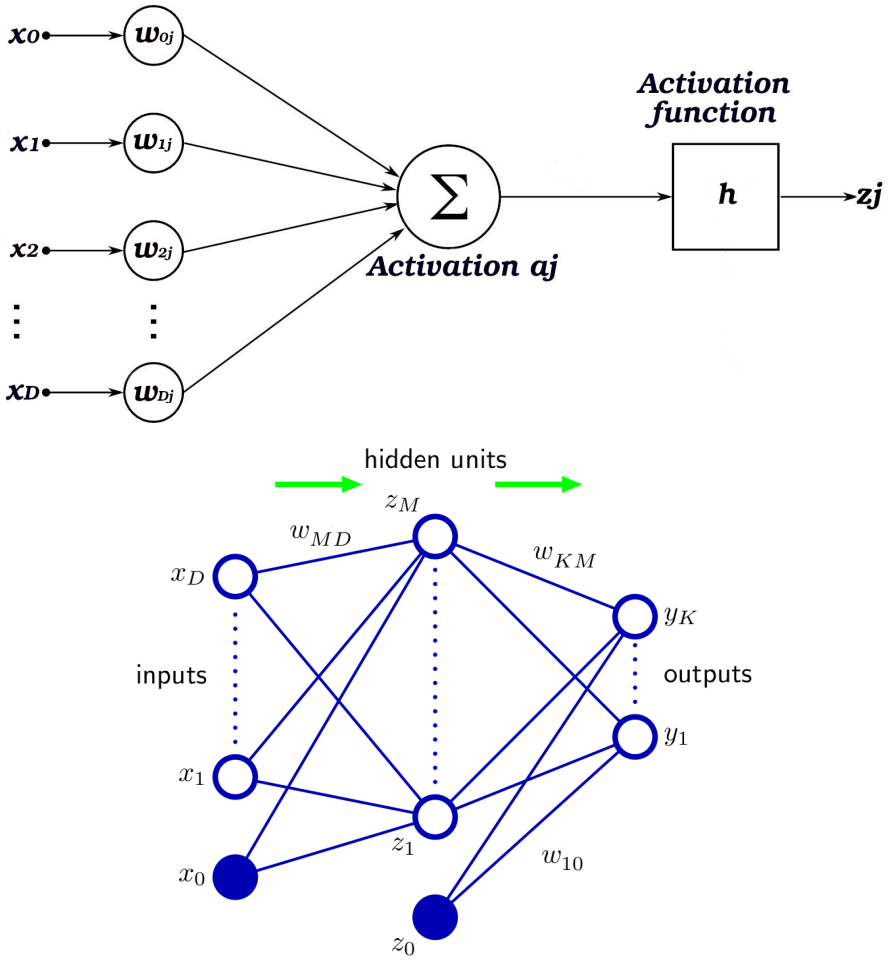

Figure 2.7: An illustration of an artificial neuron and an artificial neural network (ANN, Bishop 2006). Top: an artificial neuron receives D inputs and uses an activation function h to generate an output z_j, where $j \in [0, M]$. Bottom: an ANN with D inputs and k final outputs. Each final output neuron receives M inputs and applies an activation function, usually a sigmoid function, to give a final output value.

input data helps the model to learn complex patterns from the training examples. The bottom panel of Figure 2.7 shows the final outputs y_k, which are obtained by combining the outputs z_j from the previous layer (Bishop, 2006):

$$y_k(\mathbf{x},\mathbf{w}) = h\left(\sum_{j=1}^{M} w_{kj}\; h\left(\sum_{i=1}^{D} w_{ji}x_i + w_{j0}\right) + w_{k0}\right). \tag{2.9}$$

There are various forms of activation functions that can be used when building an ANN model (Yuen et al., 2020):

- Sigmoid function:

$$h(x) = \frac{1}{1 + e^{-x}} \ , \tag{2.10}$$

- Hyperbolic tangent function (Tanh):

$$h(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \ , \tag{2.11}$$

- Softmax function:

$$h(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \ , \tag{2.12}$$

- Rectified Linear Units (ReLU) function:

$$h(x) = \begin{cases} 0, & \text{if x} < 0 \\ 1, & \text{if x} \geq 0 \end{cases} \ . \tag{2.13}$$

The choice of these activation functions depends on the nature of the problems to be solved. For example, sigmoid is used for binary classification, while softmax is well-suited for multi-class classification. These functions are usually applied at the output layer of the network, while ReLU is used in the hidden layers. The weights and biases are optimised during the so-called backpropagation phase of the model training.

2.3.2 Backpropagation algorithm

Backpropagation (Rumelhart et al., 1986) is an algorithm used by neural networks to optimise the weights and biases of each neuron during the training process. This algorithm implements a technique known as gradient descent to reduce the training error E between the real target values and the network output values. The training error is computed by using a loss function which estimates the loss score to be minimised. A basic form of the loss function is the sum-of-squares error, which is associated with weight vectors $\mathbf{w}$ defined in the so-called *hypothesis space* (Mitchell, 1997):

$$E(\mathbf{w}) = \frac{1}{2} \sum_{d \in D} (y_d - o_d)^2 \ , \tag{2.14}$$

where D is a set of training examples, d is an instance of the examples with a target value y_d, and o_d is the output value produced by the network. A simple example of hypothesis space is illustrated in Figure 2.8, for a particular neuron with two weights w_0 and w_1. As shown in the figure, the error $E(w)$ fluctuates as a function of the combination of w_0 and w_1. The gradient descent finds the optimal values of these two weights that best minimise $E(w)$. The algorithm first initialises these two parameters with random values, then computes the gradient of $E(w)$, and finally updates the weight values by a small factor, known as *learning rate* (η), until it reaches a global minimum error. For each iteration, the weights are updated

as:

$$w_i \leftarrow w_i + \Delta w_i \ ,$$

$$\text{with} \qquad \Delta w_i = \eta \frac{\partial E}{\partial w_i} \qquad \text{or} \qquad \Delta w_i = \eta \sum_{d \in D} (y_d - o_d)^2 x_{id} \ , \tag{2.15}$$

where x_{id} is an input value associated with the target y_d. In practice, the gradient descent has to run multiple steps before finding the local minima. In addition, searching for the global minimum is hard if there are many local minima on the error surface. An alternative approach called stochastic gradient descent has been developed to alleviate these issues. This approach accelerates the searching process and keeps the algorithm from falling into the local minima (Ruder, 2016). The stochastic gradient descent algorithm updates the weights incrementally as it computes the error for each training example (Equation 2.16), instead of updating the weights after summing the errors from all training sets as shown in the previous equation. Therefore, Δw_i becomes:

$$\Delta w_i = \eta (y - o) x_i \ , \tag{2.16}$$

where y is the target value, o is the output, and x_i is an instance of the training example. The backpropagation algorithm updates each network weight by applying gradient descent

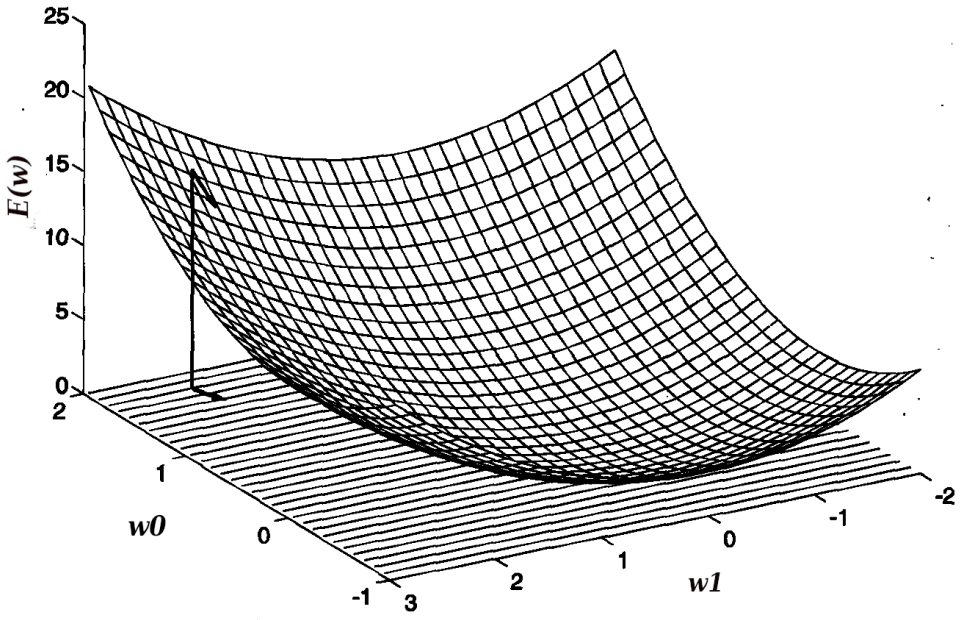

Figure 2.8: This figure shows a simple example of a hypothesis space with two weights w_0 and w_1 (Mitchell, 1997). This space is defined by all possible values of the error function $E(w)$ with respect to the two weight values. The gradient descent algorithm searches through this space for a combination of w_0 and w_1 that best minimises $E(w)$.

backward through the network from the bottom layer to the top layer.

2.4 Generative adversarial networks

Genarative adversarial networks (GANs) are neural networks that belong to the class of generative models in machine learning. The concept of GANs was introduced by Goodfellow et al. (2014) to generate new synthetic examples for the MNIST handwritten digit dataset (Lecun et al., 1998), the CIFAR-10 tiny images dataset (Krizhevsky, 2009), and the Toronto Face Database (Susskind et al., 2010). Since then, many variants of GAN models have been developed and have shown impressive results, such as the Deep Convolutional GAN (DCGAN, Radford et al. 2015), the Information Maximizing GAN (InfoGAN, Chen et al. 2016), and BigGAN (Brock et al., 2018). The key differences between these types of GANs are found in their loss functions and network architectures (Wang et al., 2019b). GANs have been used in diverse applications, especially in the field of computer vision, including image-to-image translation (Isola et al., 2016), text-to-image translation (Zhang et al., 2016), and image generation of human faces (Karras et al., 2017). More recently, the application of GANs have started to gain attention in the field of Astronomy. For instance, GANs have been implemented to generate synthetic samples for pulsar candidate classification (Guo et al., 2019). A GAN model consists of a combination of two neural networks: a generator and a discriminator. These two networks have a distinct role and goal: the generator is trained to create synthetic images, while the discriminator is trained as a classifier. The generator aims to generate realistic fake images, whereas the discriminator classifies between fake and real images. The term *adversarial* refers to the fact that these two models are trained together competitively; the generator is trained to create realistic images which are hard for the discriminator to classify, and the discriminator is trained to discriminate between fake and real images.

2.4.1 The generator model

The generator model aims to produce images with a probability distribution as close as possible to the probability distribution of the original training examples. Mathematically, given a random input noise vector z drawn from a random probability distribution $p_z(z)$ (also known as latent space), the generator model outputs fake images with a random probability distribution p_g through a neural network denoted as $G(z;\theta_g)$, where θ_g represent model parameters. The network is trained on training examples $\mathbf{x}$ with a probability distribution $p_{data}(x)$. The generated samples are represented as $G(z)$. The objective of the generator is to generate images with $p_g(x)$ as close as possible to $p_{data}(x)$.

2.4.2 The discriminator model

The discriminator is a classifier model which takes as input variables the original training data, with a probability distribution $p_{data}(x)$, and the generated samples, with a probability distribution $p_g(x)$. The discriminator model is a neural network denoted as $D(x;\theta_d)$ with

parameters θ_d, which outputs a single scalar $D(x)$, if the input data is from real samples, or $D(G(z))$ if the data is from generated samples. These two outputs have values $\in [0, 1]$.

2.4.3 GAN loss function

The original formulation of GAN uses the so-called *Minimax GAN loss* function, which refers to the optimisation of both the discriminator and generator models simultaneously (Goodfellow et al., 2014). This version of GAN is also known as *Vanilla* GAN. Besides, various forms of GAN loss functions have been developed since Vanilla GAN was introduced, such as the Least Squares GAN Loss (Mao et al., 2016) and Wasserstein Loss (Arjovsky et al., 2017). This section describes the Minimax loss function, which is derived by applying a binary cross-entropy loss function to the discriminator and the generator models. The binary cross-entropy loss is used for a binary classification task, which is ideal for classifying fake and real samples in GAN and is described as:

$$L(\hat{y}, y) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}), \tag{2.17}$$

where $\hat{y}$ and y are the model output prediction and the real target value respectively.

For the discriminator model, the output prediction is $\hat{y} = D(x)$ if the data comes from real examples, where the class label $y = 1$. Therefore, Equation 2.17 becomes:

$$L(D(x), 1) = \log(D(x)). \tag{2.18}$$

On the other hand, if the input data of the discriminator are from the fake examples, then the output $\hat{y} = D(G(z))$, with $y = 0$. Thus, Equation 2.17 gives:

$$L(D(G(z), 0) = \log(1 - D(G(z))). \tag{2.19}$$

As stated above, the goal of the discriminator is to classify correctly sample data from the real sample and fake sample. To achieve this goal, Equation 2.18 and 2.19 have to be maximised since the discriminator model can accurately predict an input example if $D(x) = 1$ and $D(G(z)) = 0$. As a result, maximising these expressions can be written as :

$$\max_D \; L(D) = \max_D \; \{\log(D(x)) + \log(1 - D(G(z)))\}. \tag{2.20}$$

For the generator model, $\hat{y} = D(G(z))$ and $y = 0$. Again, the generator model goal is to create realistic images so that $D(G(z)) = 1$. Thus, the generator is only trained to minimise Equation 2.19:

$$\min_G \; L(G) = \min_G \; \{\log(D(x)) + \log(1 - D(G(z))\}. \tag{2.21}$$

The term $\log(D(x))$ in this equation has no effect in minimising $log(1 - D(G(z)))$. It is included as a purpose of generalisation. Furthermore, the above equations are only for one

instance of the training examples. To generalise the loss function over all examples, one needs to take the expectation of these equations, where the minimax loss function is expressed as (Goodfellow et al., 2014):

$$\min_G \max_D V(D,G) = \min_G \max_D \Big\{ \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \Big\}. \quad (2.22)$$

This equation implies that the generator is trained to maximise the probability that the discriminator makes a mistake, while the discriminator is trained to do the opposite. The minimum loss is achieved when the minimax loss reaches a point called the Nash equilibrium (Nash, 1951) in which the discriminator and generator losses are both at their minimum value (Goodfellow et al., 2014). In other words, this is also the point where the optimal discriminator is obtained. Almost all extensions of GANs focus on searching for the convergence of GAN training to the Nash equilibrium (e.g., Kodali et al. 2017).

2.4.4 Training a GAN algorithm

Training a GAN model is a challenging task because optimising either the discriminator or the generator performance can worsen the performance of the other (Goodfellow, 2016). Finding the Nash equilibrium is then difficult in GANs, resulting in unstable GANs training. As a result, it remains an ongoing research in the field of machine learning, with the primary goals of improving the stability of GAN training and obtaining realistic synthetic samples (Denton et al. 2015; Jiwoong Im et al. 2016; Salimans et al. 2016). The two primary steps in training a Vanilla GAN algorithm are as follows: The first step is to train the discriminator on a set of real and synthetic examples while fixing the generator weights. In other words, this step trains the discriminator to classify between fake and real examples. The second stage of training is to use a batch of generated examples to train the generator model to create more realistic examples. Here, the weights of the discriminator are fixed, and the batch of generated examples is feed-forward to the discriminator via a composite model that connects the generator and the discriminator. The discriminator then computes the difference between the fake and real examples and backpropagates the training error to the generator. The latter uses the error to update its weights so that it can generate more realistic examples in the next training iteration. The Vanilla GAN training algorithm uses a minibatch stochastic gradient to update the weights of the discriminator and generator networks, as summarised in Algorithm 1 and illustrated in Figure 2.9 (Goodfellow et al., 2014). In recent GAN frameworks, Adam optimisation algorithm (Kingma & Ba, 2014) has gained popularity in comparison to other optimisation algorithms used in GANs. Adam is a gradient descent extension inspired by the Adaptive Gradient Descent (AdaGrad) and Root Mean Square Propagation (RMSProp) algorithms. Several advantages of using the Adam optimiser include its ease of implementation, computational efficiency, and low memory requirement.

Algorithm 1: Training algorithm for Vanilla GAN using minibatch stochastic gradient descent.

Initialization: define the generator $G(z;\theta_g)$ and the discriminator $D(x;\theta_d)$.

for N number of training iterations **do**

for k batches **do**

A) Fix the generator weights and train D.

• Sample a mini-batch of m noise samples $\left\{z^{(1)},...,z^{m}\right\}$ from a random probability distribution $\mathrm{p}_z(z)$.

• Sample a mini-batch of m real training examples $\left\{x^{(1)},...,x^{m}\right\}$ from the data distribution $\mathrm{p}_{data(x)}$.

• Update the Discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}\left[logD(x^{(i)}+log(1-D(G(z^{(i)})))\right]$$

B) Fix the weights of D and train G.

• Sample a mini-batch of m noise samples $\left\{z^{(1)},...,z^{m}\right\}$ from a random probability distribution $\mathrm{p}_z(z)$.

• Update the Generator by descending its stochastic gradient.

$$\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\left[log(1-D(G(z^{(i)})))\right]$$

end

end

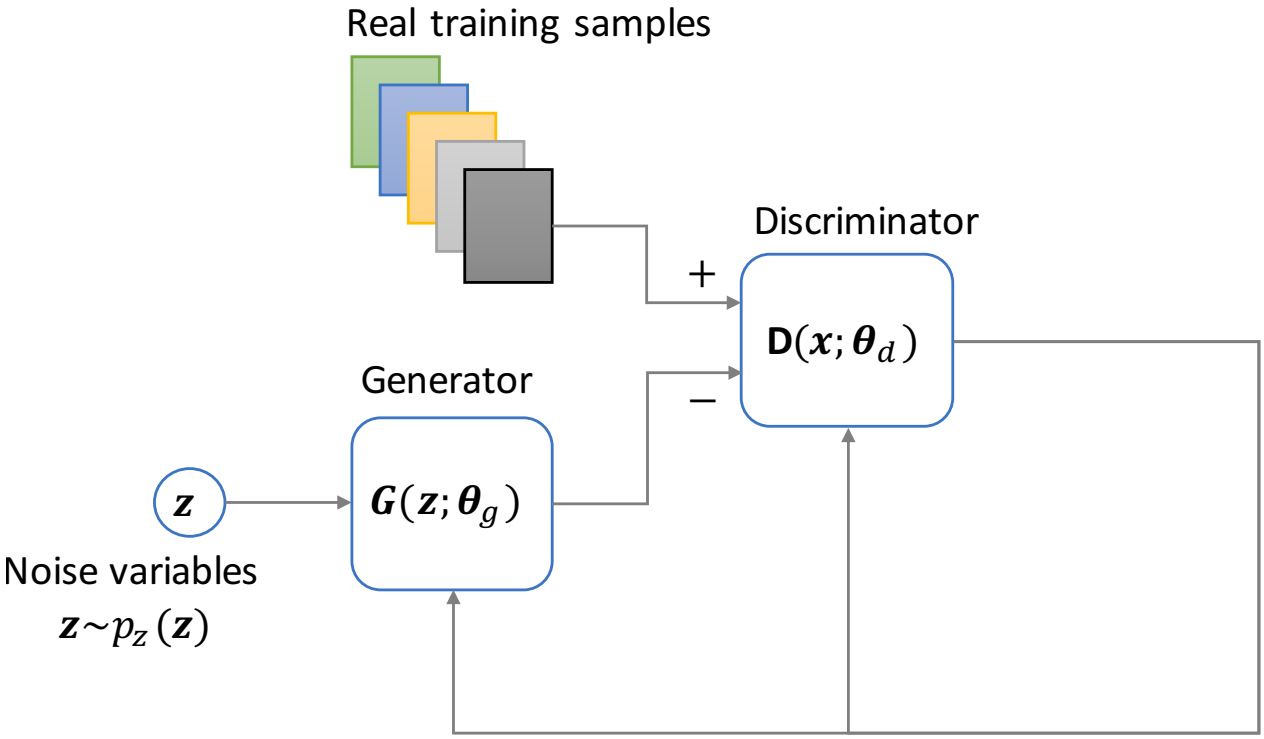

Figure 2.9: A visual representation of a Vanilla GAN training algorithm (Guo et al., 2019). The generator $G(z;\theta_g)$ generates fake examples from random noise variables z. The discriminator $D(x;\theta_d)$ is trained with a batch of fake and real examples to minimise the probability of classifying fake examples as real examples. The discriminator updates its weights via backpropagation. While the discriminator weights are fixed, the generator is trained with a batch of fake examples. The generator receives feedback from the discriminator and updates its weights through backpropagation.

2.4.5 Evaluation metrics

Different evaluation metrics are used to evaluate and compare our models. These metrics are derived based on the so-called *confusion matrix* which is a table-based representation of a model's performance. Each value in the table represents the number of correct or incorrect predictions made by the model:

- True Positive (TP): it refers to the number of predicting a positive class correctly.
- True Negative: it refers to the number of predicting a negative class correctly.
- False Positive (FP): it refers to the number of incorrectly predicting a negative class as positive.
- False Negative (FN): it refers to the number of incorrectly predicting a positive class as negative.

Table 2.1 shows a confusion matrix for a binary classification problem. To evaluate the performance of SGAN and Bad-GAN, the following evaluation metrics are adopted for a binary classification task:

- Accuracy provides the proportion of correct predictions among the total number of predictions. This metric is ideally used when both negative and positive class labels are equally important and well-balanced:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \ . \tag{2.23}$$

- Precision determines what proportion of predicted positives are truly positive. It is used when high precision is required for positive label prediction:

$$Precision = \frac{TP}{TP + FP} \ . \tag{2.24}$$

- Recall determines what proportion of true positives are correctly predicted:

$$Recall = \frac{TP}{TP + FN} \ . \tag{2.25}$$

- F-score: this metric keeps the balance between recall and precision score. A high F-score means high recall and precision scores. This metric is used when positive labels are more important. For example, we want to be sure if predicted pulsars are truly pulsars (precision) and we want to correctly predict as many true pulsars as possible (recall):

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \ . \tag{2.26}$$

- Specificity is the inverse of the recall score. It measures the proportion of true negatives correctly predicted:

$$Specificity = \frac{TN}{TN + FP} \ . \tag{2.27}$$

- Geometric mean (G-mean) is ideally used when both negative and positive labels are equally important, and if the dataset exhibits highly imbalanced classes:

$$G - mean = \sqrt{2 \times Recall \times Specificity} \ . \tag{2.28}$$

- False positive rate (FPR) is similar to the precision score. It measures the proportion of negative labels incorrectly predicted as positive. It is usually used in highly imbalanced class datasets, where positive class labels are important. Note that a lower value of FPR corresponds to better performance:

$$FPR = \frac{FP}{FP + TN} \ . \tag{2.29}$$

Table 2.1: Confusion matrix for a binary classification task. Positive and Negative correspond to Pulsar and Non-pulsar respectively.

		Predicted label	
		Positive	Negative
True label	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Negative (FP)	True Negative (TN)

2.5 Summary

This chapter introduced the concepts of machine learning and deep learning, which are intended to solve complex problems such as image classification and pattern recognition. We discussed three distinct approaches to machine learning: supervised, semi-supervised, and unsupervised learning. Supervised learning is dedicated to solving machine learning tasks when labelled data is available; semi-supervised learning is used to solve machine learning tasks when there are few labelled and a large amount of unlabelled data; and unsupervised learning is used when we want the algorithm to learn on its own without labelled data. Since this thesis focuses on the classification of pulsar candidates using both supervised and semi-supervised approaches, we elaborated on various machine learning algorithms that can be used for classification tasks, such as Random Forest and XGBoost. Additionally, we discussed a deep learning model capable of addressing complex tasks more effectively than a standard machine learning algorithm: the artificial neural network. Finally, we discussed GANs, a type of generative model that is used to create realistic synthetic images. This chapter also covered different evaluation metrics for evaluating the performance of machine learning algorithms. The following chapters will focus on the application of the machine learning techniques described in this chapter to the classification of pulsar candidates.

Chapter 3

Supervised learning for pulsar classification

3.1 Introduction

The use of machine learning (ML) algorithms has become widespread in the search for pulsars, with the goal of reducing the number of candidates that must be visually inspected. Among the many applications of ML, the most frequent use for candidate classification is that of supervised learning (Mitchell, 1997) introduced in Section 2.1.1. In pulsar astronomy, Eatough et al. (2010) pioneered the development of a supervised machine learning method for classifying pulsar candidates from Parkes Multibeam Pulsar Survey (Manchester et al., 2001) using ANNs. They were able to recover 92% of known pulsars from about 2.5 million pulsar candidates in their testing set using 8 feature variables. They concluded that one reason that pulsars from the test sample may have been missed is because the ANNs were poorly trained on millisecond pulsars, another was that their search engine created abnormal candidate plots, and the third was because the training sets were imbalanced. Bates et al. (2012) attempted to replicate the latter's work using data from the HTRU survey (see Section 1.4.4) in order to improve the performance of the ANN model by increasing the number of features to 22. Nevertheless, this study did not find a significant difference in the model's performance when increasing the number of features. In addition, they suggested that further improvements to the classification techniques are required in order to detect more pulsars.

Later, Morello et al. (2014) developed a pipeline called Straightforward Pulsar Identification using Neural Networks (SPINN) to classify pulsar candidates in the HTRU-S survey (see Section 1.4.4). They proposed that a sufficient number of pulsar samples is required in order to train a neural network model with a high number of features; otherwise, the model performance becomes worse. Using only six features, their model achieves a maximum recall score of 100% with a false positive rate of 0.64%, and a minimum false positive rate of 0.01% with a recall score of 95%. The observed increase in the model performance in their work

could be attributed to the use of a less complex model with a minimal number of features.

Recent studies focused on searching for feature selection techniques and exploring various machine learning algorithms to minimise false positive rate and recover more pulsars: Lyon et al. (2016) built a tree-based algorithm, the Gaussian Hellinger Very Fast Decision Tree (GH-VFDT), that was designed for on-line sifting of pulsar candidates. Combined with a new set of eight features, their approach was able to recover more than 90% of pulsars from millions of candidates. Tan et al. (2018) worked on LOTASS pulsar candidates and introduced new pulsar features to improve ML classification performance using an ensemble learning approach; Bethapudi & Desai (2018) tested various machine learning techniques, such as ANNs and Adaboost, using the HTRU-S dataset and a data oversampling technique to deal with imbalanced data; Wang et al. (2019a) implemented several ML algorithms, including XGBoost, Random Forest, and Hybrid Ensemble, to classify pulsar candidates from HTRU1 and HTRU2 datasets (see Section 3.2). They also applied *feature relevant importance* to select features prior to the training process. A similar study has been carried out by Lin et al. (2020a). Although these studies have shown impressive results, it is worth investigating other features and feature selection techniques (Lyon et al. 2016; Wang et al. 2019a), and testing the algorithms/features on different datasets (Tan et al. 2018; Bethapudi & Desai 2018, which could bring improvements in machine learning classification performances in future pulsar surveys.

The main purpose of this study is to develop a machine learning approach that is as efficient as possible in detecting pulsars while minimising contamination from noise candidates. To achieve this goal, we investigate ML techniques such as feature selection and hyper-parameters optimisation, while also comparing different ML algorithms.

3.2 Data used for this study

The datasets used in this work are, namely, the HTRU1 and HTRU2 datasets. The HTRU1 dataset is the output of the new processing of HTRU Medium Latitude (medlat) data (Keith et al., 2010) which was reprocessed by Morello et al. (2014) using a GPU-based pipeline known as PEASOUP (Barr, 2020). Morello et al. (2014) created the dataset for machine learning purposes by manually processing the pulsar candidates. This dataset contains 1196 real pulsars and 89,996 non-pulsars containing both RFI and other sources of noise. In addition, Morello et al. (2014) made this dataset publicly available for the creation and testing of various machine learning algorithms by the larger scientific community. The HTRU2 dataset, on the other hand, was created by Lyon et al. (2016) through an analysis of HTRU-medlat data by Thornton (2013). It contains 1639 real pulsars and 16,259 non-pulsars. Before training, the data is scaled to have zero mean and unit variance. For each dataset, we report the performance of each ML algorithm on the original data and on the re-sampled data using

various data sampling methods described in Section 3.5 to address the skewed distribution of the data. Besides, before proceeding with any model training, the data is split into training and testing sets. The testing set is only used to test the performance of a final model and is not involved in any training or feature selection. The testing set accounts for 40% and 20% of the entire dataset for HTRU1 and HTRU2 respectively. The proportion of the testing set is consistent with that of Wang et al. (2019a) and Guo et al. (2017) for comparison.

3.3 Methodology: machine learning classifiers

3.3.1 Feature selection

Many features have been proposed to characterise pulsar candidates, and some of them have been found to be useful (e.g., Morello et al. 2014; Lin et al. 2020a). For HTRU1 and HTRU2 datasets, 8 features from Lyon et al. (2016) and 22 features from Bates et al. (2012) are commonly used in the feature selection process (e.g., Wang et al. 2019a; Lin et al. 2020a). The details of these features can be found in Table 3.1. Each feature is indexed with a numerical index (f_{ID}). The eight features from Lyon et al. (2016) have been proven to be effective and have no intrinsic biases. However, they were only derived from the folded profile and the DM-SNR curve (see Section 4.3.1 and Figure 4.1), which may have omitted certain information. Therefore, in this work, we combine these features with those from Bates et al. (2012) and applied feature selection techniques to select the most relevant features.

3.3.2 Software packages

The software packages used in this work are based on Python 3.8.5 and are described as follows: the Random Forest classifier was implemented using *RandomForestClassifier()* from the sklearn library (version 0.23.2); the XGBoost classifier is implemented using XGBClassifier() imported from the XGBoost library version 1.4.2; the ANN classifier was built using the Keras library (version 2.4.3); Bayesian optimisations are performed using the Hyperopt library (version 0.2.5). Data samplings are carried out using the EditedNearestNeighbours and SMOTEENN functions from *imbalanced-learn* python tools (version 0.8.0). The codes developed by me and used in this thesis can be found on GitHub[1].

3.3.3 Feature selection strategies

We select features based on tree models which output feature relative importance. It reflects the contribution of different features to the performance of the tree algorithms. As discussed in Section 2.2.4 and 2.2.5, Random Forest is a tree-based algorithm which uses Gini impurity or Information gain/entropy to output the feature importance. When a tree is being trained, the algorithm determines how much impurity is reduced by each feature. The more impurity

[1] `https://github.com/rtprincy/MPhil-project`

Table 3.1: Details of the 30 combined features used for feature selection.

Feature ID	Feature description
	Lyon et al. (2016)
f_1	Mean of the integrated (folded) pulse profile.
f_2	Standard deviation of the integrated (folded) pulse profile.
f_3	Skewness of the integrated (folded) pulse profile.
f_4	Excess kurtosis of the integrated (folded) pulse profile.
f_5	Mean of the DM-SNR curve.
f_6	Standard deviation of the DM-SNR curve.
f_7	Skewness of the DM-SNR curve.
f_8	Excess kurtosis of the DM-SNR curve.
	Bates et al. (2012)
f_9	Chi-Squared value of a sine curve fit to pulse profile.
f_{10}	Chi-Squared value of a sine-squared curve fit to pulse profile.
f_{11}	Number of peaks in the profile -1.
f_{12}	The sum of the residuals.
f_{13}	Distance between expectation values of Gaussian and fixed Gaussian fits to profile histogram.
f_{14}	Ratio between max values of Gaussian and fixed Gaussian fits, to profile histogram.
f_{15}	Distance between expectation values of derivative histogram and profile histogram.
f_{16}	Full-width-half-maximum of a Gaussian fit to the pulse profile.
f_{17}	Chi-squared value of a Gaussian fit to the pulse profile.
f_{18}	Smallest FWHM of a double Gaussian fit to the pulse profile.
f_{19}	Chi-squared value of a double Gaussian fit to the pulse profile.
f_{20}	Best candidate period.
f_{21}	Best Candidate S/N.
f_{22}	Best candidate DM.
f_{23}	Candidate duty cycle.
f_{24}	$SNR/\sqrt{(P-W)/W}$.
f_{25}	$SNR_{fit}/\sqrt{(P-W)/W}$.
f_{26}	$mod(DM_{fit} - DM_{best})$.
f_{27}	Chi squared value from DM curve fit.
f_{28}	RMS of peak positions in all sub-bands.
f_{29}	Average correlation coefficient for each pair of sub-bands.
f_{30}	Sum of correlation coefficients.

reduction a feature achieves, the more important it becomes. The final feature importance is obtained by the averaged feature importance across all trees in the forest. This operation can be done in Random Forest by computing the so-called mean decrease impurity (Breiman et al., 1984) which provides the total decrease in impurity averaged across the trees.

In XGBoost, feature scores are computed based on the number of times a feature is used to split each node in a decision tree. The more frequently a feature is used in the decision tree to make critical decisions, the higher its score (Hastie et al., 2009). The feature importance scores are obtained by averaging each feature score across all decision trees in the XGBoost classifier. Following the work of Wang et al. (2019b), the training set is used to train Random Forest and XGBoost to obtain the feature importance scores. It is worth noting that no optimisation is made at this stage for both models. The training is run for 100 iterations as each iteration yields slightly different feature importance scores for Random Forest. XGBoost produces the same feature importance scores for all training iterations, thus there are no error bars in Figure 3.1. This figure indicates that the feature f_3 which is the skewness of the integrated pulse profile is the most important feature for both XGBoost and Random Forest and for both datasets. This is followed by f_9 (Chi-Squared value of a

sine curve fit to pulse profile) and f_{18} (smallest FWHM of a double Gaussian fit to the pulse profile) for Random Forest for both datasets. In the case of XGBoost, f_{22} (best candidate DM) and f_9 (Chi-Squared value of a sine curve fit to pulse profile) are the second and third most important features for HTRU1, and f_9 and f_{20} (best candidate period) for HTRU2. Furthermore, XGBoost feature scores exhibit a skewed distribution for HTRU1 and HTRU2,

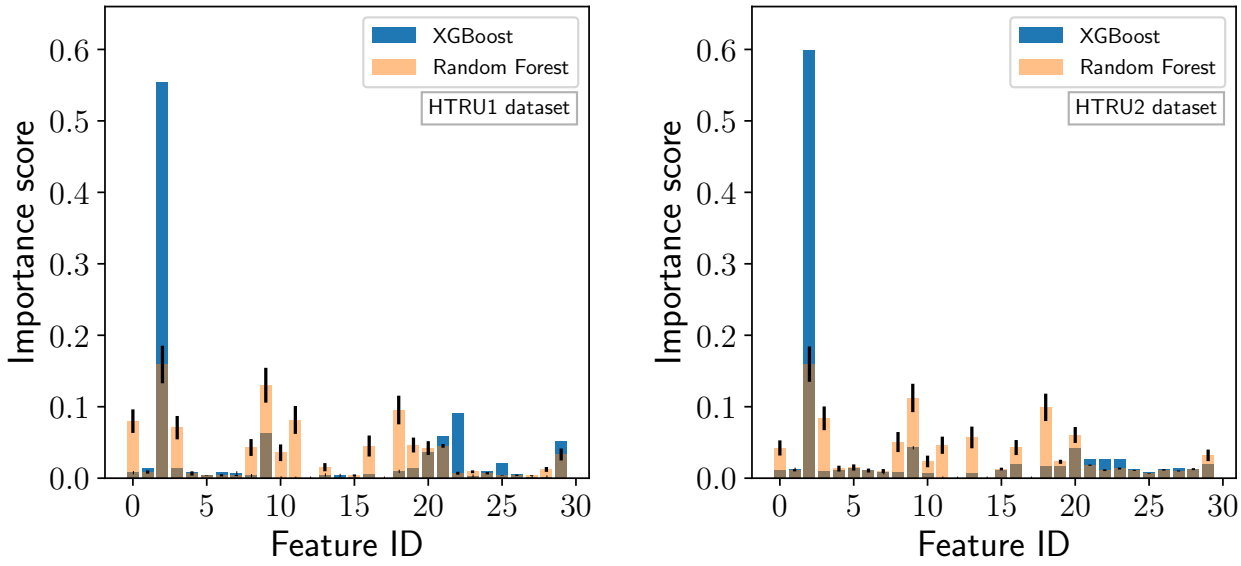

Figure 3.1: Feature importance scores from XGBoost (blue) and Random Forest (orange) for HTRU1 (left) and HTRU2 (right). The importance score varies between 0 and 1. High importance score indicates how important a feature is for the classification problem.

which may imply that few features are enough for XGBoost to separate pulsars from non-pulsars. On the other hand, Random Forest feature scores are normally distributed, which may imply that more features are required to distinguish pulsars from non-pulsars for Random Forest. Besides, the findings of Lin et al. (2020a) concluded that 4 features are sufficient for boosting models to achieve an optimised performance, while Random Forest requires 6 features to achieve the same performance. In this work, we use the eight most important features selected by Random Forest and XGBoost to train three classifiers: Random Forest, XGBoost, and ANN. The lists of these features are shown in Table 3.2 and 3.3.

3.3.4 Model optimisation with Bayesian tree-structured Parzen Estimator

Most machine learning algorithm implementations offer parameters that users can adjust, each of which alters the performance of a selected model. These parameters are known as *hyper-parameters* of the model. Hyper-parameter optimisation is the process of screening through the space of possible parameter values for a ML algorithm to identify a set of parameter values that enables the algorithm to produce more desirable results. In practice, hyper-parameters have a substantial effect on the success of ML algorithms. There are several

Table 3.2: A list of the eight most important features selected by Random Forest based on feature importance score.

Common selected features for HTRU1 and HTRU2
Skewness of the integrated (folded) pulse profile
Chi-Squared value of a sine-squared curve fit to pulse profile
Chi-squared value of a double Gaussian fit to the pulse profile
Excess kurtosis of the integrated (folded) pulse profile
The sum of the residuals
HTRU1 features
Mean of the integrated (folded) pulse profile
Best candidate period
Best candidate DM
HTRU2 features
Best Candidate S/N
Ratio between max values of Gaussian and fixed Gaussian fits, to profile histogram
Chi-Squared value of a sine curve fit to pulse profile

Table 3.3: A list of the eight most important features selected by XGBoost based on feature importance score.

Common selected features for HTRU1 and HTRU2
Skewness of the integrated (folded) pulse profile
Chi-Squared value of a sine-squared curve fit to pulse profile
Best Candidate S/N
Best candidate DM
Candidate duty cycle
Sum of correlation coefficients
HTRU1 features
$mod(DM_{fit} - DM_{best})$
Best candidate period
HTRU2 features
$SNR/\sqrt{(P-W)/W}$
Chi-squared value of a Gaussian fit to the pulse profile

techniques for optimising hyperparameters, including Random Search (RS, Bergstra & Bengio 2012) and Grid Search (Larochelle et al. 2007). It is suggested that Sequential Model-Based Optimisation (SMBO), also known as Bayesian optimisation, is one of the most effective approaches for objective function minimisation (Bergstra et al., 2015). Some of the advantages of this technique are that it can handle continuous, discrete, and conditional hyper-parameters; it can also handle parallel evaluations of an objective function with hundreds of hyper-parameters. It is therefore well-suited for optimising the hyper-parameters of most machine learning algorithms. A Python module called *Hyperopt* was developed by Bergstra

et al. (2013) to implement the Bayesian optimisation algorithm and is used for optimising XGBoost and Random Forest hyper-parameters in this work. However, other optimisation algorithms such as the RS can also be used interchangeably with the Bayesian optimisation in Hyperopt (Bergstra et al., 2015). With Hyperopt, hyper-parameters/configuration space can be defined as a probability distribution in which users can flexibly define a range of plausible values for each hyper-parameter. Hyperopt provides some configurations, via a function called $fmin$, that need to be defined prior to the optimisation process:

- Objective function

This function is one of the arguments of $fmin$ and is referred to as q hereafter. The role of this function is to take as an argument the hyper-parameter space that stands for *param* (defined below) and return a single loss value to be minimised. An objective function is defined inside the function q to compute this loss. This objective function receives *param* as an argument and the loss is computed by means of an evaluation metric. In this work, RandomForestClassifier and XGBClassifier functions are used as objective functions for Random Forest and XGBoost respectively. For Random Forest, a Sklearn function known as *cross_val_score* (cross-validation score) is used to compute the loss. A $5 - fold$ cross-validation, a F1-score metric (suitable for binary classification that requires balanced precision and recall scores), and the training set along with its corresponding target are the arguments of the *cross_val_score* function. The cross-validation results in five scores and the loss to be minimised is derived as mean(1−F1-score). This loss is the output of the q function. In the case of XGBoost, a native XGBoost API cross-validation function (xgboost.cv) is used. It has some arguments including the training set and its corresponding target, the number of k-fold cross-validations, and the metrics to be used. Here, a 5-fold cross-validation and a F1-Score metric are used. The loss is computed as mean(1−F1-score).

- Hyper-parameter space

The hyper-parameter space specifies the domain where Hyperopt may scan. This space is a Python dictionary referred to as *param*. The keys and values of this dictionary are the hyper-parameters of the objective function and their corresponding range values. For example, in XGBoost, to search for an optimal value of learning rate (ranging from 0.001 to 0.1) and the number of estimators (from 100 to 200 with a step size of 10), the hyper-parameter space *param* is expressed as follows:

```
from hyperopt import hp
from hyperopt.pyll import scope

param = {
'learning_rate': hp.uniform('learning_rate', 0.001, 0.1),
'n_estimators': hp.quniform('n_estimators', 100,200,10))
        }
```

The function hp.uniform is used to draw continuous values uniformly between a two-sided interval, while hp.quniform is used for parameters that take a discrete value.

- Search algorithm

A search algorithm defines a hyper-parameter optimisation method that is used to search for optimal hyper-parameter values that minimise the objective function. Hyperopt provides two popular search algorithms, including the RS and Bayesian Tree Parzen Estimators (TPE, Bergstra et al. 2011). In this work, the TPE approach, which is one of the SMBO variants, is adopted since it has been shown to yield promising results in several studies (e.g., Wang & Ni 2019; Thornton et al. 2013). It can be simply implemented in Hyperopt via *hyperopt.tpe.suggest* as an argument of the $fmin$ function. To run the Hyperopt algorithm, the Bayesian TPE optimisation is implemented as follows:

```
trials=Trials()
best=fmin(fn=Objectives,
    space=param,
    algo=hyperopt.tpe.suggest,
    max_evals=300,
    trials=trials)
```

The parameter max_evals in $fmin$ defines the number of times the objective function is evaluated. Each iteration corresponds to one hyper-parameter search. For each evaluation, Hyperopt uses the Trials() object to record the losses and hyper-parameters values.

3.3.5 Implementation of the Bayesian optimisation

As discussed in Section 3.3.4, a parameter space contains the hyper-parameters of a model or classifier and their possible values. The parameter space used in this work for XGBoost and Random Forest are summarised in Table 3.4 and Table 3.5 respectively. The lower and upper limits of each hyper-parameter space are chosen so that the hyper-parameter default value is in the middle or near the space limits. Besides, some other hyper-parameters have been left to their default value and they are not presented in the tables. These default values either have no significant effect on the classifier's performance (in the case of the XGBoost classifier, Wang & Ni 2019) or are already the best values for the hyper-parameters (for Random Forest). To obtain the optimal values in Table 3.4 and Table 3.5, the $fmin$ function is run with a maximum evaluation (max_evals) of 300 for XGBoost and Random Forest for both the HTRU1 and HTRU2 datasets. Initially, Hyperopt selects uniformly hyper-parameter values from the whole range of the parameter space, and as the iteration increases, the algorithm learns more and starts focusing on a narrow range of the best values, as shown in Figure 3.2. The maximum evaluation value is chosen because at 300 iterations, the hyper-parameters are found in this narrow range of values.

Table 3.4: XGBoost Hyper-parameters space and optimal values for HTRU1 and HTRU2 datasets.

		Optimal value	
Hyper-parameter	Space	HTRU1	HTRU2
learning_rate	(0.001,0.5)	0.3463	0.3942
Subsample	(0,1)	0.9113	0.2972
colsample_by_tree	(0,0.5)	0.4457	0.4165
Gamma	(0.01, 0.25)	0.0615	0.1522
max_depth	(3,12)	10	3
min_child_weight	(5,15)	13	7

Table 3.5: Random Forest Hyper-parameters space and optimal values for HTRU1 and HTRU2 datasets.

		Optimal value	
Hyper-parameter	Space	HTRU1	HTRU2
max_depth	(15,25)	17	20
min_samples_split	(5,10)	6	6
min_samples_leaf	(1,5)	2	2
min_weight_fraction_leaf	(0,0.0005)	0	0.0003
min_impurity_decrease	(0,0.002)	0	0
Criterion	(gini,entropy)	entropy	entropy

Regarding the optimisation speed, XGBoost optimised the hyper-parameters more than 10 times faster than Random Forest, with an optimisation time of 1.22sec/evaluation and 15.11sec/evaluation for the HTRU1 dataset for XGBoost and Random Forest respectively (processed on 14 CPU cores @2.2GHz with 512 GB memory machine).

3.3.6 Optimisation of the ANN model

The hyper-parameters of the ANN model are optimised using a 5-fold cross validation. It consists of splitting the training data into 5 stratified partitions using the StratifiedKFold Sklearn function. The cross-validation starts by using 4 partitions to train the model, while 1 partition is used to evaluate (as a validation set) the model's performance. The next iteration used another partition different from the previous one to evaluate the model, while the rest are used as a training set. These processes continue until each partition is used as a validation set. At the end of the cross-validation, five scores are obtained. Here, F1-score is used to evaluate the model. These five scores are averaged and the averaged score is used as the model performance for the specified hyper-parameters. These parameters include the number of neural network layers, the number of neurons at each layer, learning-rate, activation functions (ReLU, Sigmoid, Tanh), optimiser algorithms (RMSprop, Adam), and batch size. These values are tuned during the optimisation processes and the values that yield the best performance score are used to train the final model. The best parameters are summarised in Table 3.6. The configuration 8:8:1 in this table means that the model has one

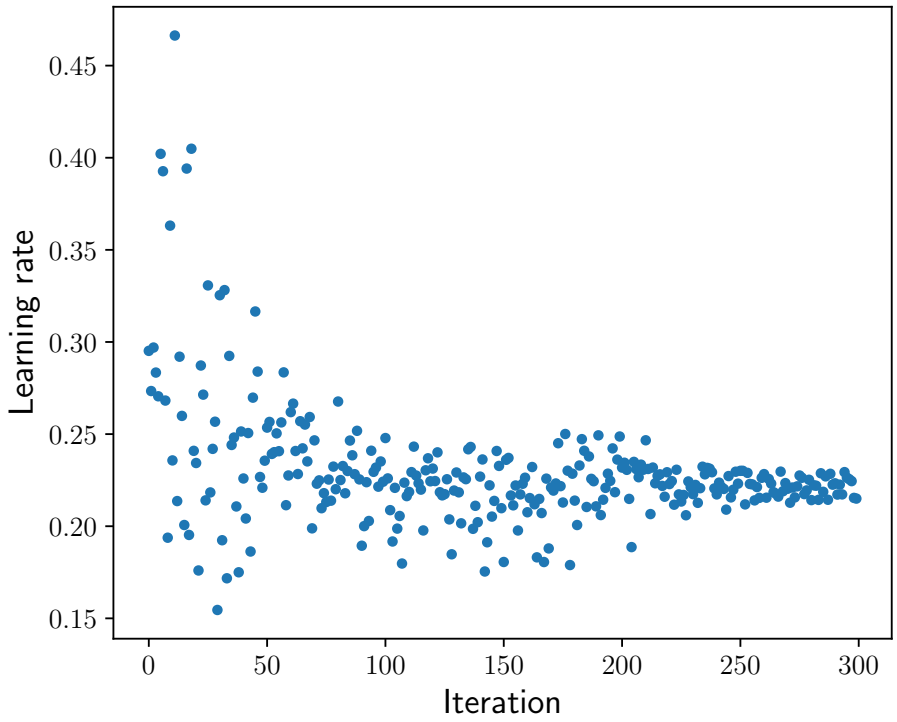

Figure 3.2: Illustration of optimisation of the learning-rate for 300 iterations (evaluations). At the beginning of the iteration, the learning-rate values are spread across the range of possible values. As the number of iterations increases, the learning-rate starts focusing on the best value range.

input layer with 8 input features, one hidden layer with 8 neurons, and one output layer with 1 output. Besides, a ReLU function is used in the hidden layer, while a Sigmoid function is applied in the output layer. Note that the eight selected features from XGBoost are used during the optimisation procedure as these features produced good performance on the cross-validation score compared to those selected from Random Forest. However, for comparison purposes, both sets of features are used to train the ANN model. The performance scores are shown in Table 3.7 and 3.8.

Table 3.6: ANN optimal hyper-parameters for HTRU1 and HTRU2 datasets. The model has one input layer with 8 input features, one hidden layer with eight neurons, and one output layer with 1 output. ReLU and Sigmoid activation functions are used in the hidden and output layers respectively.

Hyper-parameter	HTRU1	HTRU2
Network	8:8:1	8:8:1
Activation function	ReLU/Sigmoid	ReLU/Sigmoid
Optimiser	Adam	Adam
Learning-rate	0.001	0.001
Batch size	128	100

3.3.7 Model implementation and evaluation: Random Forest, XGBoost, and Artificial Neural Networks

This section focuses on the implementation and evaluation of three types of ML algorithms used in this chapter: Random Forest, XGBoost, and Artificial Neural Networks. These algorithms have been described in the previous chapter.

Random Forest: the optimised Random Forest model is trained on the training set using the selected features in Table 3.2. Although this classifier produced low variance results (a slightly different performance score for each time the classifier is trained) as described in Section 2.2.3, it is trained five times on the same training data. The median values of the performance scores are reported in Table 3.7 and Table 3.8. Note that the median of values is chosen here, and throughout the thesis, instead of the mean. The median of data points is frequently used since it is less affected by data outliers than the mean (Boslaugh, 2012).

XGBoost is trained in the same manner as Random Forest, expect that it is trained on the selected features in Table 3.3. Unlike Random Forest, XGBoost always produces the same performance scores each time it is trained on the same training data.

ANN: the ANN model is implemented using a Python library Keras. The eight features selected by Random Forest and XGBoost are used to train the optimised model. The ANN is trained for 500 epochs using a Keras function *EarlyStopping* to prevent the model from overfitting. Using this function, the training could stop before reaching 500 epochs once the model performance on the validation set stops improving (the validation loss stops decreasing). Furthermore, ANN produced a different performance score each time it is trained, thus the model is trained five times and the median value of the scores is reported. The results of the evaluations are presented in Section 3.3.8.

In this work, various evaluation metrics (see Section 2.4.5) are used to report the results. However, we focus on F1-score and specificity metrics when comparing and discussing the results. The first metric is used when we want the classifier to correctly predict all true pulsars present in the dataset (corresponds to recall score), while assuring that all candidates predicted as pulsars are truly pulsars (corresponds to precision score). For instance, an F1-score of 1 implies that all real pulsars in the testing set are correctly predicted as real pulsars (Recall=1), and none of the predicted pulsars are non-pulsars (precision=1). The specificity, on the other hand, measures the proportion of true negatives predicted correctly in the testing set. A specificity of 1 means that all non-pulsars in the testing set is correctly predicted by the classifier. This may be useful in pulsar searches as all non-pulsars predicted could be discarded from the analysis, which reduces the number of pulsar candidates to be inspected by pulsar experts.

3.3.8 Results and discussions

After training each classifier, they are evaluated on the same testing set using the evaluation metrics described in Section 2.4.5. Our results show that the performance of the classifiers is different when they are trained on the HTRU1 and HTRU2 datasets. Following the same classification procedures on both datasets, such as feature selection and model optimisation, the classifiers trained on HTRU1 perform better than those trained on HTRU2 in all metric scores.

For HTRU1, XGBoost yields the best F1-score of 0.970. It can be interpreted as follows: from the 35999 negative and 478 positive candidates in the testing set, XGBoost predicted 460 candidates from them as positives, where 455 of them are recovered from the 478 candidates (which gives the recall score of about 0.952) and that 455 of the 460 are true positives (a precision of about 0.989). Random Forest, on the other hand, produced the best specificity of 0.9999 (approximated as 1.000 in Table 3.7), where 35,996 of the 35,999 negative candidates are correctly predicted. In other words, only 3 true negative candidates in the testing set are not recovered. In comparison with the work of Wang et al. (2019a), they used the same sizes of training and testing sets as in this work and trained the XGBoost classifier with 27 of the features in Table 3.1 (excluding f_{13}, f_{11}, and f_{15}), which resulted in an F1-score of 0.977. Although the classifiers performed adequately on HTRU1, a data sampling technique is applied to the training data in Section 3.5 to recover more pulsars in the testing set, which might further improve the F1-score.

For HTRU2, the classification scores in Table 3.8 reveal that the recall scores are lower than the other scores for all classifiers, which in turn decreases the F1-score performance of the classifiers. This is most likely caused by the overlapping features of non-pulsars and pulsars in the feature space, as shown in Figure 3.3. Most negative candidates lie in the overlapping area, and the classifiers might predict most positive candidates in this area as negative candidates. As a result, the classifiers could not recall most of the positive candidates. Under-sampling the negative candidates in the training set in this area would increase the Recall score. However, negative candidates in the testing set that are located in the overlapping area could be predicted as positive candidates in this case. This in turn lowers the precision score and increases the false positive rate. To confirm this hypothesis, an under-sampling technique is applied to the HTRU2 dataset, which is described in Section 3.5.2.

Regarding the performance of the classifiers on HTRU2, XGBoost produced the best F1-score and Specificity of 0.919 and 0.996 respectively. From 3,252 negative and 328 positive candidates in the testing dataset, XGBoost predicted 305 of them as positive candidates, where it recovered 291 true positives out of the 328 positive candidates (giving a recall of

0.887) and that 291 of the 305 predicted as positives are true positives (giving a precision of 0.954). The specificity, on the other hand, provided insight into the proportion of the true negative candidates recovered in the prediction: XGBoost is able to recover 3,238 negative candidates out of the 3,252 true negatives in the testing set. In comparison, Wang et al. (2019a) obtained an F1-score of 0.907 using an XGBoost classifier trained on the 30 features presented in Table 3.1

Regarding the performance of ANN on the selected features, Table 3.7 shows that ANN performed better on the eight features selected by XGBoost compared to those selected by Random Forest. This is true for both HTRU1 and HTRU2, where the difference is more significant in HTRU2. A possible explanation for these results may be the feature selection strategy used by XGBoost. The latter scores a feature based on the number of times it is used in decision trees (as discussed in Section 3.3.3), where each tree is dependent on the prior tree. This allows the classifier to select only the most frequently used features in the trees rather than computing individual feature scores in each non-correlated tree, as in Random Forest (one important feature in one tree might be important in other trees). Also, this result may explain the distribution of the feature importance scores in Figure 3.1 and 3.4, where the most important feature selected by XGBoost has a score higher than 50% while the other 29 features contribute to the rest. These findings further support the conclusion of Lin et al. (2020a) that more features are required to train a Random Forest classifier to achieve the same performance of XGBoost classifier with a few features.

Table 3.7: Performance of different methods on HTRU1 dataset. $ANN^{(1)}$ and $ANN^{(2)}$ correspond to the ANNs trained on the eight selected features by XGBoost and Random Forest respectively.

Classifier	**Recall**	**Precision**	**F1-Score**	**FPR** [%]	**Accuracy**	**G-Mean**	**Specificity**
XGBoost	0.952	0.989	0.970	0.014	0.999	0.976	1.000
RF	0.916 ± 0.003	0.993 ± 0.001	0.954 ± 0.001	0.008 ± 0.001	0.999 ± 0.000	0.957 ± 0.002	1.000 ± 0.000
$ANN^{(1)}$	0.954 ± 0.005	0.975 ± 0.006	0.968 ± 0.004	0.033 ± 0.007	0.999 ± 0.000	0.977 ± 0.002	1.000 ± 0.000
$ANN^{(2)}$	0.944 ± 0.005	0.970 ± 0.004	0.957 ± 0.003	0.039 ± 0.005	0.999 ± 0.000	0.971 ± 0.002	1.000 ± 0.000

Table 3.8: Performance of different classifiers on HTRU2 dataset. $ANN^{(1)}$ and $ANN^{(2)}$ correspond to the ANNs trained on the eight selected features by XGBoost and Random Forest respectively.

Classifier	**Recall**	**Precision**	**F1-Score**	**FPR** [%]	**Accuracy**	**G-Mean**	**Specificity**
XGBoost	0.887	0.954	0.919	0.400	0.986	0.940	0.996
RF	0.851 ± 0.003	0.921 ± 0.003	0.884 ± 0.002	0.700 ± 0.040	0.980 ± 0.000	0.919 ± 0.002	0.993 ± 0.000
$ANN^{(1)}$	0.875 ± 0.006	0.941 ± 0.007	0.907 ± 0.006	0.553 ± 0.067	0.984 ± 0.001	0.933 ± 0.003	0.994 ± 0.001
$ANN^{(2)}$	0.838 ± 0.007	0.917 ± 0.003	0.879 ± 0.004	0.768 ± 0.031	0.979 ± 0.001	0.912 ± 0.004	0.992 ± 0.000

3.4 How to deal with an imbalanced dataset

Our datasets are dominated by non-pulsar candidates, especially for the HTRU1 dataset. As a result, the datasets are imbalanced with a ratio of 1:75 and 1:10 for HTRU1 and HTRU2 respectively. This skewed distribution has been shown to introduce a bias in a model and lower its performance (Lyon et al., 2016). To address this problem, we apply popular sampling techniques to balance the datasets by either under-sampling the majority class with Edited Nearest Neighbors, (ENN, Wilson 1972), or over-sampling the majority class with Synthetic Minority Over-sampling Technique (SMOTE, Chawla et al. 2011), or combining the over-sampling and under-sampling techniques with SMOTE and ENN, referred to as SMOTEEN (Batista et al., 2004). The SMOTE technique has been used before by Lin et al. (2020a) and Bethapudi & Desai (2018), and has shown good results for pulsar candidate classification in the HTRU1 dataset.

3.5 Methodology: data sampling

3.5.1 SMOTE

One approach to alleviate the imbalanced class is to over-sample the minority class. SMOTE is a widely-used method to synthesise new examples. It works by choosing random examples in the feature space, drawing a line between them, and then creating a new sample along that line. More precisely, a random example from the minority class is selected first. Then the algorithm searches for the k-nearest neighbours to that example. In the feature space, a synthetic example is produced at a random location between the first selected example and its random nearest neighbour. SMOTE is combined with an under-sampling technique described in Section 3.5.3.

3.5.2 Edited Nearest Neighbors rule

Another approach to alleviate the imbalanced class datasets is to under-sample the majority class. The ENN rule consists of removing ambiguous and noisy examples in a dataset that are misclassified by their 3-nearest neighbours (Wilson, 1972). The ENN algorithm works as follows: the three closest neighbours of each example X in the dataset are calculated. If X is a majority class example and its three closest neighbours misclassify it, X is deleted from the dataset. If X is a minority class example and its three closest neighbours misclassify it, these majority class examples are deleted from the dataset.

ENN is used to under-sample the majority class in the HTRU2 dataset with the aim to increase the Recall score as discussed in Section 3.3.8. In this work, the ENN rule is implemented using the EditedNearestNeighbours imbalanced-learn Python class. Two arguments are defined in ENN: *sampling_strategy* and *n_neighbors* (n-nearest neighbours); "majority" is used as sampling strategy so that only candidates in the majority class misclassified by "50"

nearest-neighbours are removed. The performance of each classifier on the under-sampled datasets is reported in Section 3.5.5.

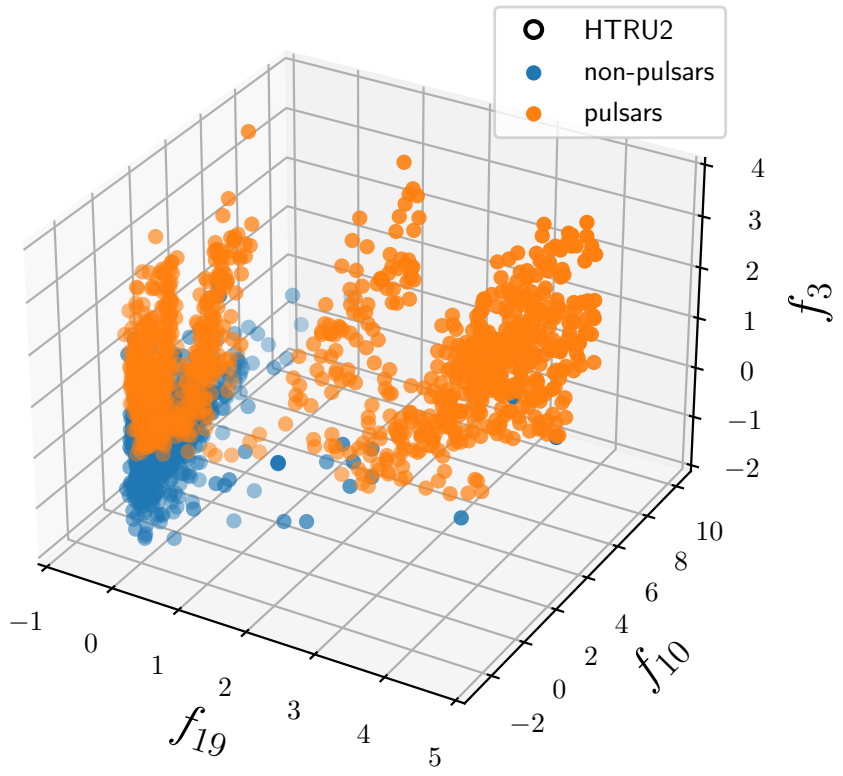

Figure 3.3: A 3-D plot of the three most important features commonly selected by XGBoost and Random Forest. The overlapping of the negative and positive candidates in this figure could explain the classifiers' low performance scores for the HTRU2 dataset.

3.5.3 SMOTEENN

SMOTEEN is one of the under-sampling and over-sampling combination techniques developed by Batista et al. (2004) when they explored different combination techniques, including Condensed Nearest Neighbors + Tomek Links, SMOTE + Tomek Links, and SMOTE + ENN. This study suggested that the two last methods are efficient for a dataset with a small number of positive minority examples. Besides, they found that ENN provides more in-depth data cleaning than Tomek Links. Since the HTRU1 dataset is highly imbalanced (1:75) compared to the HTRU2 dataset (1:10), increasing the minority class would increase the performance of the classifiers on HTRU1. However, combining over-sampling and under-sampling (removing the majority class in the noisy area of the feature space) could also improve the performance of the classifiers. In this work, SMOTEENN is implemented to re-sample the

HTRU1 training set using the *imblearn* Python library. SMOTEEN takes SMOTE and ENN objects as arguments. A sampling strategy=1 (1:1 ratio) is also applied in SMOTEEN, which implies that SMOTE creates synthetic samples in the minority class as many as the majority class. In this implementation, SMOTE over-samples the minority class, then ENN removes misclassified examples (noises) from both classes (a negative/positive sample misclassified by its 3-nearest neighbours is removed). The resulting performance of this technique is reported in Section 3.5.5.

3.5.4 Model training

As we have discussed in the previous sections, SMOTEENN is used to balance the HTRU1 dataset, while ENN is used to balance the HTRU2 dataset. Before training, feature selection and model optimisation are carried out as in Section 3.3.3 and 3.3.5. The eight most important features selected by XGBoost and Random Forest are used to train XGBoost and Random Forest classifiers respectively, while both selected sets of eight features are used to train the ANN model as in the previous works with the original datasets. These new selected features are presented in Table 3.9 and 3.10. Besides, the optimised hyper-parameters of the three classifiers are reported in Table 3.11, 3.12, and 3.13. These values are the results of the Bayesian optimisation process using the Hyperopt library for Random Forest and XGBoost classifiers, while the ANN optimised hyper-parameters are obtained through 5-fold cross-validation as in Section 3.3.6. The same testing set that is used to evaluate the original datasets is also used to evaluate the performance of the three classifiers on the re-sampled datasets. The training procedures for each classifier are similar to the training of the original datasets.

Table 3.9: A list of the eight most important features selected by Random Forest based on feature importance score using the re-sampled training set.

Common selected features for HTRU1 and HTRU2
Skewness of the integrated (folded) pulse profile
Chi-Squared value of a sine-squared curve fit to pulse profile
Chi-Squared value of a sine curve fit to pulse profile
Chi-squared value of a double Gaussian fit to the pulse profile
Number of peaks in the profile -1
Sum of correlation coefficients
Best Candidate S/N
HTRU1 features
Skewness of the integrated (folded) pulse profile
HTRU2 features
Excess kurtosis of the integrated (folded) pulse profile

Table 3.10: A list of the eight most important features selected by XGBoost based on feature importance score using the re-sampled training set.

Common selected features for HTRU1 and HTRU2
Chi-Squared value of a sine-squared curve fit to pulse profile
Skewness of the integrated (folded) pulse profile
Best Candidate S/N
Best candidate period
HTRU1 features
Sum of correlation coefficients
Best candidate DM
Excess kurtosis of the integrated (folded) pulse profile
Excess kurtosis of the integrated (folded) pulse profile
HTRU2 features
$SNR_{fit}/\sqrt{(P-W)/W}$
Candidate duty cycle
Chi squared value from DM curve fit
Chi-squared value of a Gaussian fit to the pulse profile

Table 3.11: Random Forest Hyper-parameters space and optimal values using the re-sampled HTRU1 and HTRU2 training datasets.

		Optimal value	
Hyper-parameter	Space	HTRU1	HTRU2
max_depth	(15,25)	20	19
min_samples_split	(3,10)	6	8
min_samples_leaf	(1,5)	1	3
min_weight_fraction_leaf	(0,0.0005)	0	0.0003
min_impurity_decrease	(0,0.002)	0	0.0002
Criterion	(gini, entropy)	entropy	entropy

Table 3.12: XGBoost Hyper-parameters space and optimal values using the re-sampled HTRU1 and HTRU2 training datasets.

		Optimal value	
Hyper-parameter	Space	HTRU1	HTRU2
learning_rate	(0.001,0.5)	0.3212	0.3355
Subsample	(0,1)	0.9344	0.9457
colsample_by_tree	(0,0.5)	0.6386	0.4857
Gamma	(0.01, 0.25)	0.065	0.166
max_depth	(3,15)	12	6
min_child_weight	(5,15)	11.0	10

Random Forest: this classifier is trained on the re-sampled training set using the new selected features in Table 3.9. Also, the classifier is trained five times on the same re-sampled training data.

Table 3.13: ANN optimal Hyper-parameters for the re-sampled HTRU1 and HTRU2 datasets. The model has one input layer with 8 input features, one hidden layer with eight neurons, and one output layer with 1 output. ReLU and Sigmoid activation functions are used in the hidden and output layers respectively.

Hyper-parameter	HTRU1	HTRU2
Network	8:8:1	8:8:1
Activation function	ReLU/Sigmoid	ReLU/Sigmoid
Optimiser	Adam	Adam
Learning-rate	0.001	0.001
Batch size	128	100

XGBoost is trained only once using the selected features in Table 3.10, since it always produced the same results when trained on the same training set.

ANN: this model is trained on the set of eight selected features by Random Forest and XGBoost. A training epoch of 500 is used while applying the EarlyStopping Keras function. Like Random Forest, ANN is trained five times on the same training dataset.

The results of the candidate classifications are presented in Table 3.14 and 3.15. For Random Forest and ANN, the median value of the five classification scores was reported for each evaluation metric.

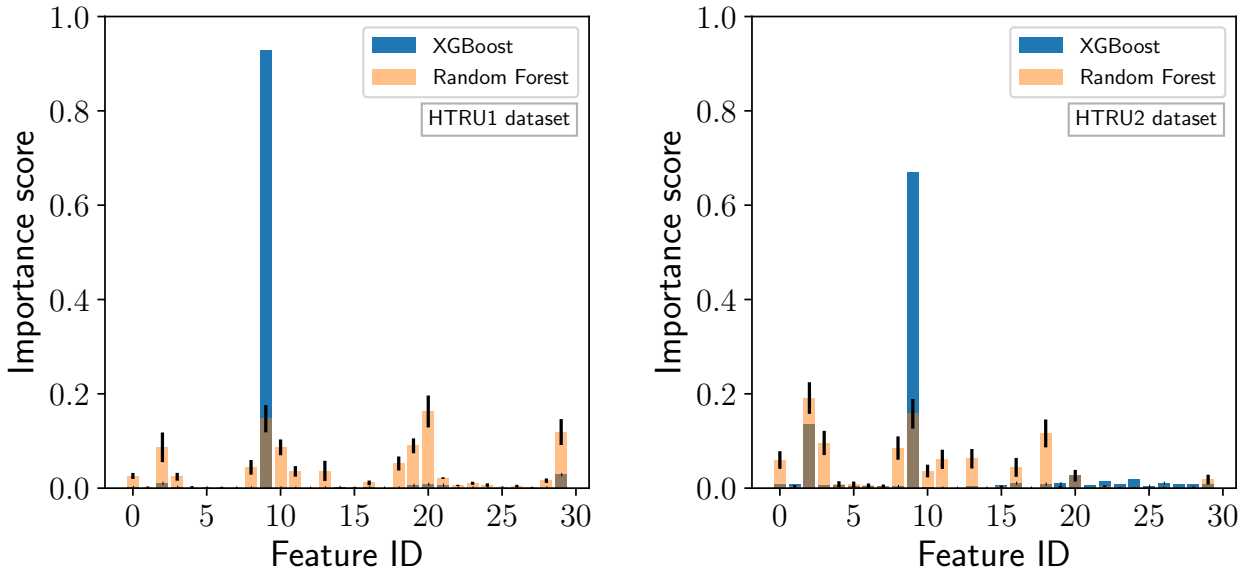

Figure 3.4: Feature importance scores from XGBoost (blue) and Random Forest (orange) for HTRU1 (left) and HTRU2 (right) using the re-sampled training datasets.

3.5.5 Results and discussions

This section presents the performance of each classifier after applying the data-sampling techniques to HTRU1 and HTRU2 datasets. These results indicate that the classifiers trained on the HTRU1 dataset outperform the performance of those trained on the HTRU2 dataset. The results in Table 3.14 and 3.15 show that the recall scores of the classifiers on both datasets increase as expected: in HTRU1, for instance, the recall score improves from 0.954 to 0.985 for ANN trained on XGBoost selected features, while in HTRU2, the recall score increases from 0.887 to 0.936 for XGBoost. However, a drop in the precision scores is observed in both datasets compared to the model performance in the original datasets, which worsens the F1-score, especially in HTRU2. A decrease in the specificity scores can also be seen in HTRU1 and HTRU2, which is very small in the case of HTRU1. These results confirm the hypothesis set in Section 3.3.8, where a drop in the precision score for classifiers trained in HTRU2 was expected.

Nevertheless, XGBoost produced the best f1-score (0.960) and specificity (0.999) on the re-sampled HTRU1 dataset with less decrease in precision and specificity. It was able to recall 468 positive candidates out of 478 positive candidates in the testing set, while it recovered only 455 positive candidates in the original dataset. Also, among the 497 predicted as positives, 468 of them are true positives: a number of a false negative of 29 against 5 in the original dataset. As for the specificity, XGBoost was able to recover 35,970 true negative candidates out of 35,999 in the testing set, while this number is 35,994 out of 35,999 in the original dataset: the 29 non-recovered became false positive candidates in the re-sampled dataset, which slightly decreased the F1-score.

Table 3.14: Performance of different classifiers on the re-sampled HTRU1 dataset. $ANN^{(1)}$ and $ANN^{(2)}$ correspond to the same models trained on the eight selected features by XGBoost and Random Forest respectively.

Classifier	**Recall**	**Precision**	**F1-Score**	**FPR** [%]	**Accuracy**	**G-Mean**	**Specificity**
XGBoost	0.979	0.942	0.960	0.080	0.999	0.989	0.999
RF	0.960 ± 0.003	0.846 ± 0.003	0.900 ± 0.002	0.233 ± 0.000	0.997 ± 0.000	0.979 ± 0.002	0.998 ± 0.000
$ANN^{(1)}$	0.985 ± 0.002	0.862 ± 0.016	0.919 ± 0.009	0.208 ± 0.028	0.998 ± 0.000	0.992 ± 0.001	0.998 ± 0.000
$ANN^{(2)}$	0.979 ± 0.005	0.654 ± 0.025	0.784 ± 0.018	0.689 ± 0.077	0.993 ± 0.001	0.986 ± 0.002	0.993 ± 0.001

Table 3.15: Performance of different classifiers on the re-sampled HTRU2 dataset. $ANN^{(1)}$ and $ANN^{(2)}$ correspond to the same models trained on the eight selected features by XGBoost and Random Forest respectively.

Classifier	**Recall**	**Precision**	**F1-Score**	**FPR** [%]	**Accuracy**	**G-Mean**	**Specificity**
XGBoost	0.936	0.764	0.841	2.921	0.968	0.953	0.971
RF	0.893 ± 0.001	0.832 ± 0.003	0.862 ± 0.002	1.814 ± 0.038	0.974 ± 0.000	0.937 ± 0.001	0.982 ± 0.000
$ANN^{(1)}$	0.881 ± 0.008	0.860 ± 0.010	0.867 ± 0.003	1.445 ± 0.138	0.975 ± 0.001	0.931 ± 0.004	0.986 ± 0.001
$ANN^{(2)}$	0.832 ± 0.004	0.893 ± 0.004	0.863 ± 0.002	1.014 ± 0.040	0.976 ± 0.000	0.908 ± 0.002	0.990 ± 0.000

Regarding the performance of the classifiers on the HTRU2 dataset, ANN outperformed the other classifiers with a F1-score of 0.867 for the ANN trained on the XGBoost selected

features and a specificity of 0.990 for the ANN trained on Random Forest selected features. However, these values are lower than those in the original dataset. Furthermore, as in the original datasets, the distribution of the feature importance scores for the re-sampled datasets (see Figure 3.4) indicates that, unlike Random Forest, XGBoost based its decision mainly on a single feature. This is most likely due to the fact that the XGBoost algorithm initially computes feature importance based on a single decision tree. Since it is an iterative algorithm, the same decision tree is used for the next iteration, which aims only at improving on the area where the previous iteration failed to classify. As a result, the best feature from the previous iteration is likely to be retained in the next iteration. In contrast, Random Forest computes feature importance scores from many independent decision trees and averages the scores from each tree to obtain the final feature importance score for each feature.

3.6 Conclusion

Supervised classification was performed on HTRU1 and HTRU2 datasets using XGBoost, Random Forest, and ANN classifiers. Feature selection was made based on Random Forest and XGBoost feature importance scores. The eight most important features were selected to train the classifiers. Then Bayesian optimisation was used to optimise the hyper-parameters of Random Forest and XGBoost, while a 5-fold cross-validation was used to optimise the ANN model. After feature selection and model optimisation, these classifiers were trained on 54715 and 14317 pulsar candidates in HTRU1 and HTRU2 respectively. A hold-out dataset (testing set) was used to evaluate the classifiers, which contains 36,477 and 3,580 pulsar candidates for HTRU1 and HTRU2 respectively. To improve the classification performance, a combination of under-sampling and over-sampling was applied to re-sample the HTRU1 dataset using SMOTEENN; whereas the majority class in HTRU2 was under-sampled using ENN. Feature selection and model optimisation followed the data re-sampling before the classifiers were trained on the re-sampled data. The main results of the classification are as follows:

1. The skewness of the integrated pulse profile is the most important feature among the 30 features selected by Random Forest and XGBoost for both the HTRU1 and HTRU2 datasets.

2. XGBoost achieved an F1-score of 0.970 and a specificity of about 1.000 on HTRU1, while Random Forest produced an F1-score of 0.954 and a specificity of 1.000 on HTRU1.

3. XGBoost produced the best F1-score and specificity on the HTRU2 dataset, with a score of 0.919 and 0.996 respectively.

4. Re-sampling the HTRU1 dataset slightly decreased the f1-score (0.960) and specificity (0.999) for XGBoost, while the recall score was increased by 2.8%.

5. Re-sampling the HTRU2 dataset increased the recall scores of the classifiers (by 5.2% for XGBoost). However, the F1-score and specificity of the classifiers dropped, especially for XGBoost. On the other hand, ANN produced the best results for the re-sampled HTRU2 dataset.

Undersampling the majority class in the HTRU2 dataset has a number of benefits and drawbacks. Although it improved the recall scores, the precision scores significantly decreased. Therefore, the f1-score for each classifier was lowered, as well as the specificity. However, this significant decrease in the precision scores might be alleviated by reducing the number of k-nearest neighbours used by ENN to reduce the number of majority class candidates to be excluded in the training data. Another alternative could be re-sampling both the minority and the majority candidates in the overlapping area, which may improve the precision scores. The drawback of this is probably that the minority candidates located in the overlapping area in the testing set might not be recalled by the classifiers. Overall, it is likely that the HTRU2 dataset is more difficult to classify than HTRU1 given the resulting performance of the classifiers on both datasets and the possible overlaps of non-pulsars and pulsars between many features. Additionally, investigating other feature selection techniques independent from the algorithms (XGBoost/Random Forest feature importance) could improve the classification performance on HTRU2.

Chapter 4

Semi-supervised GANs for pulsar classification

4.1 Overview

Semi-supervised learning plays an important role in addressing the lack of labelled data in machine learning tasks. This chapter describes and discusses the concept of GANs using semi-supervised learning to solve pulsar candidate classification tasks. A semi-supervised approach, known as SGAN, has been implemented for classifying pulsar candidates and has shown good results (Balakrishnan et al., 2021). However, Dai et al. (2017) introduced a new formulation of semi-supervised GANs called *Bad-GAN* that has been shown to produce improved model generalisation and image classification performance on three benchmark datasets: MNIST, SVHN, and CIFAR-10. The aim of this chapter is to use this method for pulsar classification tasks and compare its performance with that of the SGAN approach. The first section introduces the major challenges encountered in building a GAN model. The data used to perform the classification tasks is elaborated in Section 4.3. In addition, we discuss two types of GAN-based SSL models in Section 4.4 and 4.5, the semi-supervised GANs and Bad-GAN; and perform a comparative study between them, where the results are presented in Section 4.6.

4.2 Introduction

Classifying pulsar candidates using standard machine learning algorithms such as SVM or Random Forest is a complex process when dealing with image datasets. Before these algorithms can be trained, a process called *feature engineering* is used to manually extract relevant features from images, as investigated in Chapter 3 for the supervised classification problem. This step, however, can be omitted and performed automatically through the use of a deep learning algorithm, which saves significant time. Additionally, a sufficient amount of labelled data is required to train a supervised machine learning algorithms effectively. As

mentioned previously, acquiring this labelled data is difficult and time-consuming in pulsar astronomy, and thus a large number of pulsar candidates remain unlabelled. Semi-supervised learning has been shown to perform well with a small number of labelled data and a large number of unlabelled data. Therefore, we investigate a semi-supervised approach, notably semi-supervised GANs, to address the lack of labelled pulsar candidates.

4.3 Dataset

4.3.1 Pulsar candidate

The Pulsar candidates used in this work were observed by the High Time Resolution Universe South Low-Latitude Survey (HTRU-S Lowlat, see Section 1.4.4). Here, 84 691 preprocessed and labelled pulsar candidates described by Balakrishnan et al. (2021) are used. These candidates were produced from a re-processing pipeline of the HTRU-S Lowlat survey, using the stochastic template-bank algorithm (Harry et al., 2009), and folded with the PRESTO software (Ransom, 2011). The re-processing pipeline generated over 40 million pulsar candidates (Balakrishnan et al., 2021). A pulsar candidate consists of a four-dimensional data cube including frequency channels, time, intensity, and rotational phase (see Figure 4.1 for plot examples of these data). Four feature plots (or diagnostic plots) can be extracted from the 4D data cube and are used to visualise/classify whether a candidate is a genuine pulsar:

- DM-curve: this is a 1D plot of a quantity known as *DM trials* and its reduced-χ^2 values. "DM trials" is used to measure the dispersion of the pulsar signals from space, which is due to their propagation through the ionised interstellar medium. The DM-curve plot peaks at a non-zero value for real pulsars.
- Frequency-phase: this is a 2D image of the frequency-versus-phase plot, derived through the integration of the data over the time axis. One or more vertical lines of signals, over a broad range of frequencies, are observed in real pulsars.
- Pulse-profile: this is a 1D plot of intensity-versus-phase, obtained by integrating the data over the frequency channels and time axes. Real pulsars tend to have emissions only over a small fraction of the rotational phase, i.e., have a narrow duty cycle, and rarely have multiple widely spaced peaks.
- Time-phase: this is a 2D image obtained by integrating the data over the frequencies. In real pulsars, one or more vertical signals can be observed across all time intervals.

Each of these four images/plots is used as a training dataset for all the experiments in this work. Each of these datasets contains 84,676 labelled examples, which we further divide into labelled and unlabelled examples for the purpose of semi-supervised learning training. The labelled dataset is split in a stratified way (to have the same number of negative and positive examples) into a training set (50%), a validation set (20%), and a testing set (30%). The

Table 4.1: Number of labelled data (training set, validation set, and testing set) and unlabelled data used in this work.

Labelled dataset	Pulsar	Non-Pulsar	Total
Training set	12,914	12,914	25,828
Validation set	3,228	3,229	6,457
Testing set	6,918	6,919	13,837
Unlabelled dataset	-	-	20,000

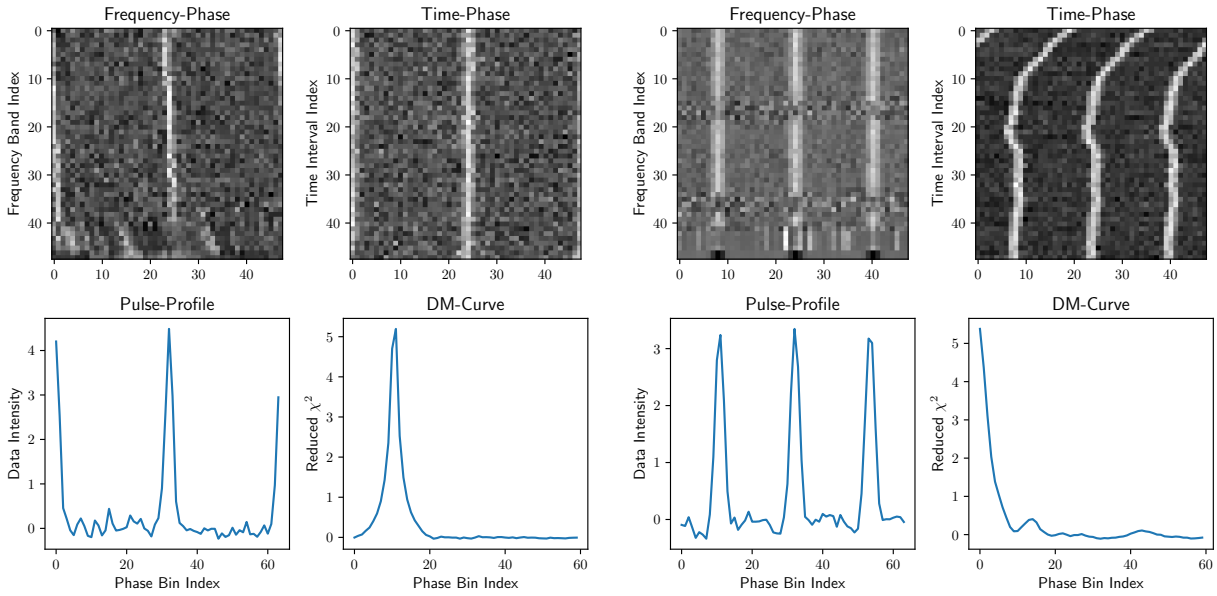

Figure 4.1: An example of a true pulsar candidate (a) and a non-pulsar candidate (b) diagnostic plots. For real pulsars, a persistent vertical stripe is expected across all frequency bands and across time intervals in the frequency-phase and time-phase plots, respectively. One narrow signal peak is observed in the pulse-profile, and DM-curve peaks at non-zero values for real pulsars

unlabelled dataset is coupled with the training set to train a semi-supervised learning model; the validation set is used to evaluate the model performance during model optimisation, and the testing set is used as a hold-out dataset to test the performance of the final model. Note that these data are normalised to have zero median and unit variance values. Regarding the dimension of each dataset, the frequency-phase and time-phase are both 48×48 pixel images, while the DM-curve and the pulse-profile are one-dimensional vectors of 60 and 64 bins respectively.

4.4 Methodology: SGAN and Bad-GAN

Semi-supervised GAN (SGAN) is an extension of GAN (see Section 2.4) that trains the discriminator to classify not only between fake and real samples, as in Vanilla GAN, but also between K classes of training samples (Odena, 2016). In SGAN, the discriminator has two components: an unsupervised part (D) that learns to classify fake and real samples and a supervised part or a classifier (C) that is trained to predict N classes of the real samples. There are several ways to implement SGANs: Radford et al. (2015) allowed D to extract features from the unlabelled dataset, then these features are reused in C to predict the classes of the real examples; Odena (2016) suggested another approach where a single discriminator model has $(K + 1)$ outputs: K outputs for C and one output for D. In this configuration, C and D share the same feature extraction layers. SGANs have been shown to improve the quality of the generated samples over the basic GAN, where only a small amount of labelled data is available. Besides, it is demonstrated that SGANs produce good performance results on a small number of labelled data when compared to a supervised learning baseline (e.g., Radford et al. 2015; Odena 2016). However, one of the most common failure modes for a

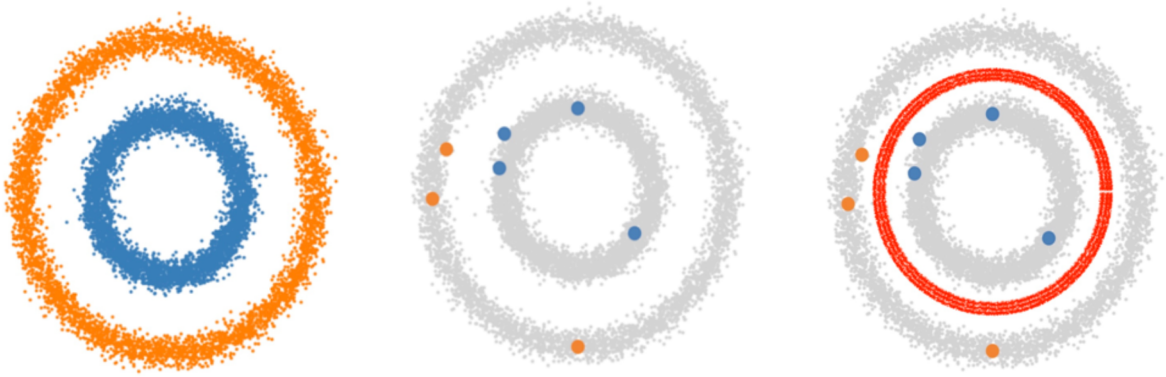

Figure 4.2: A complement generator, inspired by Petrova (2020). *Left*: the two circles represent labelled examples with two classes of training data (blues and oranges) in the feature space, where a supervised learning task is sufficient to do the classification. *Middle*: this figure corresponds to a scenario where only a few labelled training examples are available along with a large number of unlabelled examples (grey data points). A basic GAN can be used to generate samples that fill the grey area by learning from unlabelled examples. *Right*: a complement generator generates samples (in the red circles) in between the two grey circles in a Bad-GAN. These complementary samples help the discriminator to have a good decision boundary.

GAN is that the generator can collapse to a parameter setting where it always produces the same sample. To address this issue and the instability of GANs in general, Salimans et al. (2016) introduced an improved implementation of the discriminator and several techniques for training GANs, such as the so-called *feature matching* and *minibatch discrimination*.

Nevertheless, it is suggested that feature matching may not be sufficient to solve the generator mode collapse problem if the data complexity grows (Dai et al., 2017). The latter proposed a novel formulation that addresses the failures of feature matching to im-

prove the performance of SGANs. They suggest that a *complement generator* is required to achieve good semi-supervised learning. In other words, a complement generator enables a well-optimised discriminator to find good decision boundaries in the feature space. This complement generator is also known as a *bad generator* and is defined as a generator that produces sample distributions that are complementary to the existing class distributions in the feature space (See Figure 4.2). Dai et al. (2017) coined the term "Bad-GAN" to refer to this formulation of semi-supervised GAN with a complement generator.

SGAN has been applied to pulsar candidate classification (Balakrishnan et al., 2021), using a similar SGAN architecture to that described in Salimans et al. (2016). Throughout this thesis, the term "SGAN" is used to refer to the semi-supervised GAN based on the works of Balakrishnan et al. (2021). This thesis conducts a comparative study of SGAN and Bad-GAN to investigate the advantages of using Bad-GAN in pulsar candidate classification.

4.4.1 Architecture of SGAN

The architecture of a SGAN model is similar to the original GAN described in Section 2.4, except, there is a modification in the structure of the output layers of the discriminator to achieve the semi-supervised tasks in SGAN.

Here, the supervised and unsupervised discriminators are stacked in one model, where both models share the same feature layers before an activation function is applied. First, the supervised discriminator takes the output of the feature layers and applies a softmax activation function to output the K class labels of the real examples. Second, the unsupervised discriminator takes the updated feature layers and applies a custom activation function, defined as $H(x)$, to force the model to output values close to 1 for real examples and close to 0 for fake examples. The output of the custom activation function is normalised to give the output of the unsupervised discriminator (Salimans et al., 2016):

$$D(x) = \frac{H(x)}{H(x)+1} \text{ , with } H(x) = \sum_{k=1}^{K} \exp[l_k(x)] \text{ ,} \tag{4.1}$$

where l_k are known as output logits of the output layer of the classifier. The architecture of a SGAN is summarised in Figure 4.3. Convolutional neural networks (CNNs, Lecun et al. 1998) are used to construct the discriminator and generator models. CNNs are an efficient tool for pattern recognition and have been previously used for pulsar candidate selection (Zhu et al., 2014). Convolutional layers are used to extract relevant features from the input images for the discriminator and to generate synthetic images for the generator. In this work, a CNN is made up of the following layers:

- A convolutional layer: this layer convolves an input image with a smaller-size image filter, known as an image kernel. This operation returns encoded values (or feature maps) for each pattern in the input image.

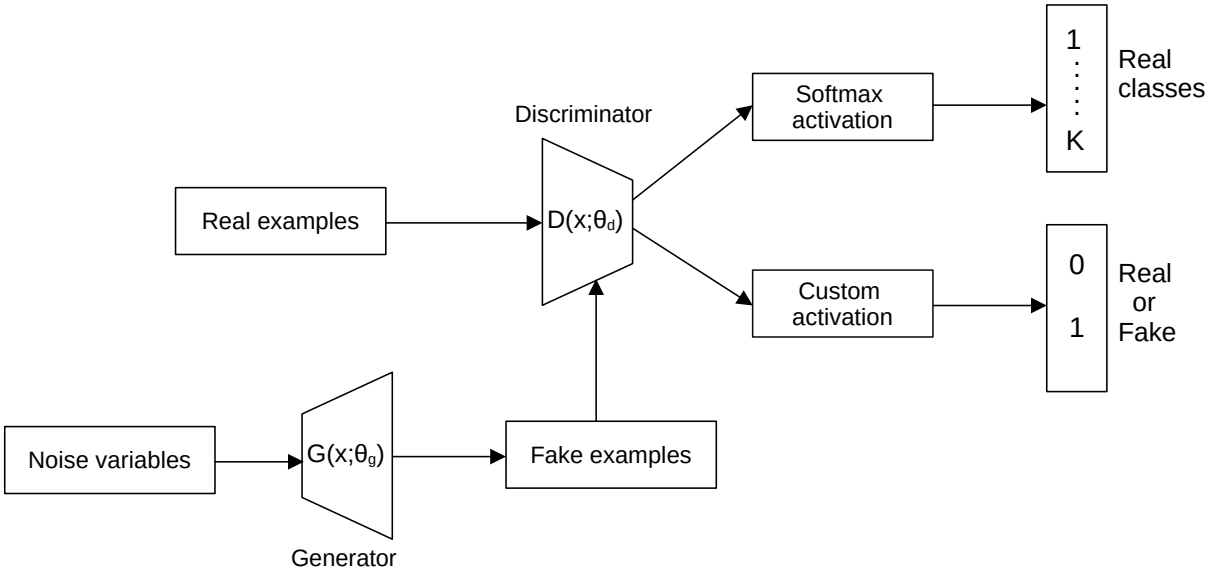

Figure 4.3: The architecture of SGAN. In this framework, the discriminator classifies between K classes of real examples, while classifying between fake and real examples. The Softmax activation and Custom activation correspond to the output of the supervised discriminator and unsupervised discriminator respectively. Both discriminators are stacked in one model, $D(x; \theta_d)$, and share the same feature layers.

- A max-pooling layer: This layer downsamples the output image from the convolutional layer. It bins the feature maps and selects only the maximum value from each bin. For instance, a max-pooling layer with a 2×2-size window downsamples a 32×32 image to a 16×16-size.
- A fully connected layer or a dense layer: this final layer flattens the outputs from the previous layers and produces a $1D$ tensor which contains the image class label.

A regularisation technique for neural networks, called *dropout rate* (Srivastava et al., 2014), is also applied to the CNN model to prevent the network from overfitting. The latter usually occurs when the model over-optimises the training data, leading the model to learn data representation/patterns specific to the training set. As a result, the model performs better on the training set than on the testing set. Overfitting is generally related to the lack of training data and the complexity of the model: in neural networks, it can be seen in a network structure with a large number of parameters (weights) or a network with a large parameter value (Bishop, 1995). Note that the dropout regularisation technique has also been shown to be effective for a less complex model trained on large datasets (Hinton et al., 2012).

4.4.2 The discriminator architecture

The discriminator architecture used in this work is illustrated in Figure 4.4. Three different architectures are built according to the size of the input image: the top panel is used for a

48×48 image input, while the other two panels correspond to $1D$ inputs with 60 and 64 bins respectively. As mentioned in the previous sections, the unsupervised and supervised models share the same feature extraction layers which end in the *Dense* layer. The Activation section in the figure has two distinct outputs: the Softmax activation function for the supervised model and the custom activation for the unsupervised one.

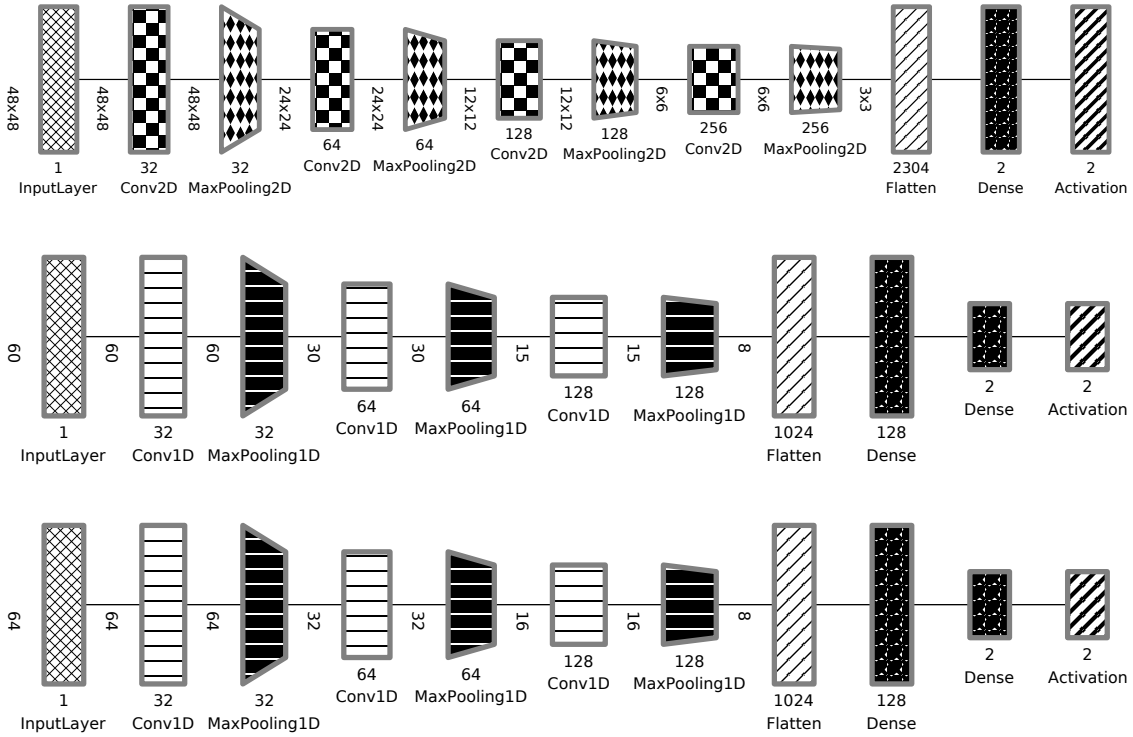

Figure 4.4: The discriminator architecture for the SGAN used in this work. The feature extraction layers end in the Dense layer for the three figures. The Activation layer represents the two activation functions (Softmax and Custom function). *Top*: a 2D image input is downsampled using a series of convolutional layers (Conv2D) to obtain a compressed representation of the image in the Dense layer. *Middle* & *bottom*: a similar approach to the top figure, but these models take 1D examples as input and use a series of 1D-convolutional layers to obtain the final outputs. The plots were built using the Net2Vis tools (Bäuerle et al., 2019).

4.4.3 The generator architecture

The generator model follows the architecture illustrated in Figure 4.5. Unlike the discriminator, the generator generates the 1D and 2D samples using a random noise vector from the latent space. It creates 48×48 images using convolution transpose layers and 60 and 64 bin samples using dense layers. Batch normalisation layers are also used to normalise the inputs of the subsequent layers (to have a mean of zero and a standard deviation of one). This technique has been shown to significantly accelerate the training process (Ioffe & Szegedy, 2015) and improve model performance, especially for convolutional networks (Goodfellow, 2016).

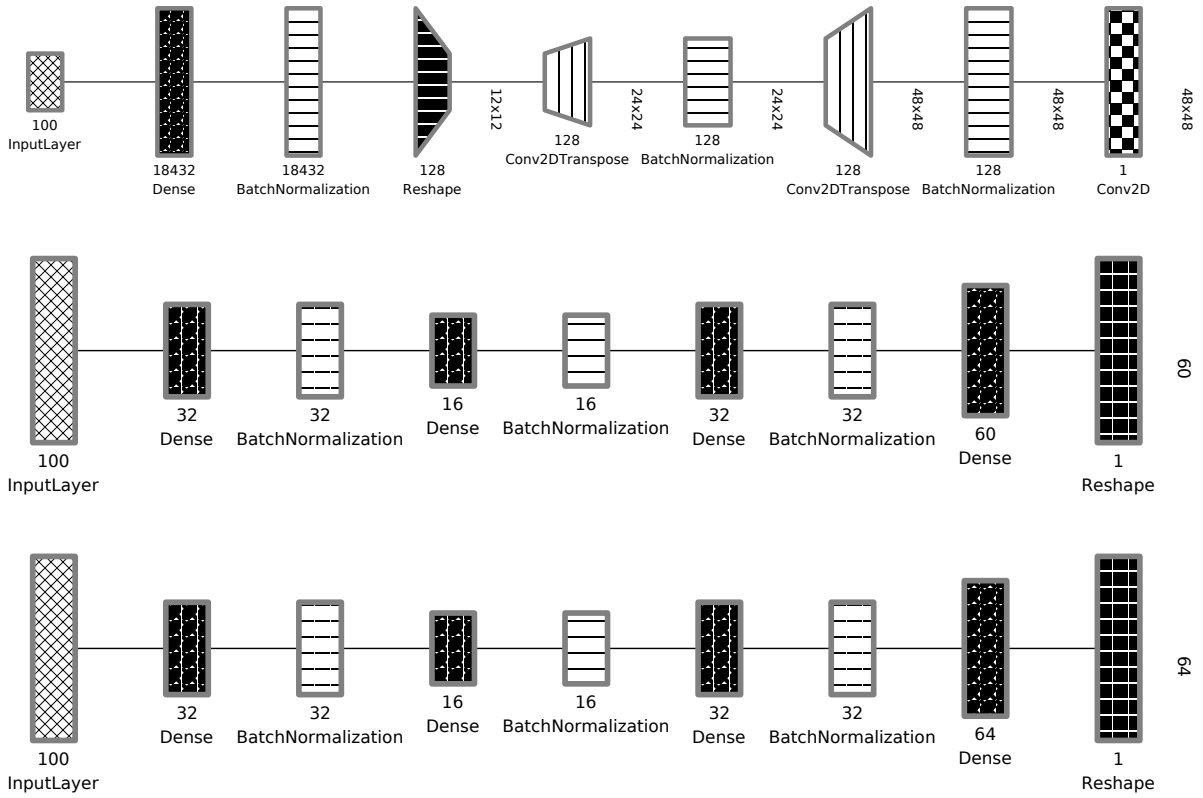

Figure 4.5: The generator architecture for SGAN. *Top*: the generator applies a series of convolutional transpose layers to the input noise vector to generate a 48×48 image. *Middle* & *bottom*: 1D samples are generated using a series of Dense layers. The plots were built using the Net2Vis tools (Bäuerle et al., 2019).

4.4.4 SGAN loss function and optimiser

The loss function of SGAN has two terms: the supervised loss and the unsupervised loss. The latter is equivalent to the loss function of the basic GAN. The supervised loss function is added to compute the loss of the supervised discriminator (Salimans et al., 2016):

$$\begin{aligned} \mathrm{L} &= \mathrm{L}_{\text{supervised}} + \mathrm{L}_{\text{unsupervised}}, \text{where} \\ \mathrm{L}_{\text{supervised}} &= -\mathbb{E}_{x,y\sim p_{data}(x,y)} \log \mathrm{D}(y|x, y < K+1), \text{and} \\ \mathrm{L}_{\text{unsupervised}} &= -\{\mathbb{E}_{x\sim p_{data}(x)} \log [1 - \mathrm{D}(y = K+1|x)] + \mathbb{E}_{z\sim p_z} \log [\mathrm{D}(y = K+1|x)]\}. \end{aligned} \tag{4.2}$$

In practice, a sparse categorical cross-entropy loss function is used to compute the loss of the supervised discriminator, while a binary cross-entropy is used for the unsupervised discriminator and the composite model (Balakrishnan et al., 2021). These loss functions are all optimised using the Adam optimiser (Kingma & Ba, 2014).

As mentioned in the previous sections, SGAN has been used by Balakrishnan et al. (2021) and has shown good classification performance with a small number of labelled data. Also, Dai et al. (2017) proposed Bad-GAN, which focuses on adding extra loss function terms to the discriminator and the generator to improve the performance of their semi-supervised GAN. For this reason, we will explore Bad-GAN in this thesis and compare its performance

with that of SGAN.

4.4.5 Bad-GAN formulation

The architecture of Bad-GAN follows the SGAN architecture presented in Figure 4.3. Also, Bad-GAN implements the same generator architecture in Figure 4.5 with slightly different discriminator outputs than that illustrated in Figure 4.4. To implement the so-called feature matching technique, the discriminator outputs an additional output, known as the feature matching layer, which is represented as a "Dense" layer in Figure 4.4. Besides, Bad-GAN adds new loss functions to the existing losses for the discriminator and the generator in Section 4.4.4. To address the collapsed generator and feature matching failures, Dai et al. (2017) introduced two additional loss function terms for the generators in addition to the feature matching loss. The first aims to increase the entropy/diversity of the generated samples in the feature space, while the second enables the generator to generate samples in low-density regions of the feature space. In Bad-GAN, the generator minimises the following loss function (Dai et al., 2017):

$$\min L_G = \min_G \{-\mathcal{H}(p_G) + \mathbb{E}_{x\sim p_G} \log p(x)\mathbb{I}[p(x) > \epsilon] + \|\mathbb{E}_{z\sim p_z} f(G(z)) - \mathbb{E}_{x\sim\mathcal{U}} f(x)\|^2\}. \quad (4.3)$$

The first term of Equation 4.3 is designed to minimise the generator mode collapse. The second term helps the generator to produce low-density samples, and the third term corresponds to the feature matching loss, which aims to generate samples closer to the unlabelled data distribution (Salimans et al., 2016). On the other hand, a conditional entropy loss term is added to the discriminator in addition to the supervised and unsupervised losses. It is suggested that this loss contributes to the success of the complement generator as well as helping the discriminator to have a correct decision boundary for unlabelled data, also referred to as true/fake belief on unlabelled data (Dai et al., 2017). In that case, the discriminator loss becomes:

$$\begin{aligned} \max L_D = \max_D \quad & \{\mathbb{E}_{x,y\sim\mathcal{L}} \log p_D(y|x, y \le K) + \mathbb{E}_{x\sim\mathcal{U}} \log p_D(y \le K|x) + \\ & \mathbb{E}_{x\sim p_G} \log p_D(K+1|x) + \mathbb{E}_{x\sim\mathcal{U}} \sum_{k=1}^{K} p_D(k|x) \log p_D(k|x)\}. \end{aligned} \quad (4.4)$$

The first three terms of Equation 4.4 are the same as the supervised and unsupervised losses in Equation 4.3, whereas the last term corresponds to the conditional entropy.

4.5 Training, prediction and evaluation metrics

4.5.1 Parameter optimisation

Before training the final Bad-GAN model, some parameters are optimised for Bad-GAN. As discussed in the previous sections, the formulation of Bad-GAN consists of implementing additional losses to the generator and the discriminator which differentiate Bad-GAN from SGAN. These losses were described in Section 4.4.5. However, we add weight terms to these losses, which are tuned during the optimisation. Since the performance of Bad-GAN is somewhat sensitive to these weights, parameter optimisation consists of finding an optimal weight for each of these loss terms. The optimisation is made by searching for the best weight value for each loss term that maximised the performance of the model on the validation set (i.e., with minimum validation loss). In practice, the weights of these losses are initially chosen based on the work of Dai et al. (2017), then they are tuned to be adapted to our models. During the optimisation, the model is trained using the training set and is evaluated using the validation set, then each parameter weight is saved with the corresponding validation loss value. Many trials are made during this process and the weights that produced the minimum validation loss are chosen. Also, the optimisation is performed for each of the datasets: DM-curve, frequency-phase, pulse-profile, time-phase. The optimised values of these weights are presented in Table 4.2 and are listed below:

Table 4.2: Optimised loss weights for Bad-GAN: *ent_weight* is the weight of the conditional entropy loss of the discriminator. The three other weights are for the generator losses: *pt_weight* is the weight of the entropy loss via pull-away term; *vi_weight* is the weight of the entropy loss via variational inference; *fm_weight* is the feature matching loss weight.

Loss weight parameters	DM-curve	Frequency-phase	Pulse-profile	Time-phase
ent_weight	0.000	0.085	0.100	0.100
pt_weight	0.100	0.800	0.100	0.100
vi_weight	0.001	0.008	0.005	0.005
fm_weight	1.000	1.000	1.000	1.000

- The conditional entropy loss weight (ent_weight): this is to balance the weight of the conditional entropy loss of the discriminator described in Section 4.4.5 (last term of Equation 4.4).
- Entropy loss via pull-away term (pt_weight) and via variational inference (vi_weight) weights. These terms were proposed by Dai et al. (2017) to maximise the entropy $\mathcal{H}(p_G)$ (Equation 4.3) of the generated examples.
- The feature matching loss weight (fm_weight), which is designed to enable the generator to generate samples closer to the unlabelled data distribution.

SGAN, on the other hand, is implemented by reproducing the works of Balakrishnan et al. (2021). Therefore, their proposed optimised parameters are used in this work for SGAN. In

Table 4.3: The hyper-parameters values of the Adam optimiser used to train SGAN and Bad-GAN. Adam offers four arguments, where two of them were kept at their default value in this work (beta2 and epsilon).

	1-D input data		2-D input data	
	Generator	Discriminator	Generator	Discriminator
Learning rate	0.0008	0.003	0.0002	0.0002
beta1	0.5	0.5	0.5	0.5

addition, the value of learning rate and batch size that are used in SGAN are also applied to Bad-GAN. Here, a batch size of 100 is used while training all datasets. The learning rates for the discriminator and the generator are summarised in Table 4.3.

4.5.2 Software/implementation

The SGAN model was implemented based on Keras (Chollet et al., 2015), an open-source Python library dedicated to building neural network models, with a Tensorflow backend (Abadi et al., 2016). On the other hand, Bad-GAN was implemented in PyTorch (Paszke et al., 2019). The details of the software versions and the implementation of SGAN and Bad-GAN on Keras and PyTorch respectively, are discussed in Appendix A.

4.5.3 SGAN training for pulsar candidate classification

The SGAN training algorithm is similar to that of the basic GAN, except that SGAN requires the training of an additional model, the supervised discriminator (or classifier). SGAN trains three models in total: a supervised discriminator, an unsupervised discriminator, and a generator. The training processes for each training iteration are as follows:

- **Supervised discriminator training**: this step involves updating the supervised discriminator. The discriminator is trained with a batch of real examples: pulsars with a positive label (1) and non-pulsars with a negative label (0).
- **Unsupervised discriminator training**: the next step is to train the unsupervised discriminator using the same feature extraction layers which were previously updated by the supervised part. Two types of losses are computed to update the unsupervised discriminator: a true loss and a fake loss. The true loss is the loss computed between a batch of unlabelled examples and a batch of real examples, while the fake loss is obtained from the loss between a batch of fake generated examples and real examples.
- **Generator training**: to train the generator, a composite model is created. This model connects the generator and the unsupervised discriminator: it receives a batch of fake examples from the generator and outputs a class label via the discriminator. In this case, the discriminator weights are fixed, and the fake examples are labelled as "1"

so that the discriminator can compute the losses/errors between these fake examples and the previously seen batch of real examples, and backpropagate these errors to the generator. The generator corrects its weights using this feedback and improves its ability to generate more realistic fake samples in the next iteration.

4.5.4 Bad-GAN training for pulsar candidate classification

The training procedures for Bad-GAN proposed by Dai et al. (2017) are adopted to train the Bad-GAN in this work. The training involves two main steps.

a) Training the discriminator

Training the discriminator consists of three steps: first, the total loss of the discriminator is computed via a loss function. Then, through a backpropagation process (see Section 2.3.2), the discriminator model is updated (weights/biases of the neural networks) and optimised using the Adam optimiser, whose hyper-parameters are presented in Table 4.3. The discriminator's total loss is computed by summing the supervised loss, the unsupervised loss, and the entropy loss (see Section 4.4.5):

- The supervised loss is the standard binary classification loss obtained by training the supervised discriminator with a batch of real examples from the labelled dataset: pulsars (1) and non-pulsars (0).
- The unsupervised loss, or true-fake loss, is the sum of true and fake losses. The true loss is the loss between a batch of unlabelled examples and real examples, while the fake loss is obtained from the loss between fake generated examples and real examples.
- The entropy loss is obtained by computing the entropy of the supervised discriminator output on a batch of unlabelled examples.

b) Training the generator

The generator model training consists of minimising the feature matching loss and the two entropy loss terms derived via variational inference and feature pull-away:

- The feature matching loss is obtained by computing the mean square difference between the feature layer output of a batch of unlabelled and a batch of fake examples.
- The entropy loss via variational inference and via pull-away term are computed using a Python class function[1] developed by Dai et al. (2017).

[1] `https://github.com/kimiyoung/ssl_bad_gan`

4.5.5 Implementation of SGAN and Bad-GAN

To conduct a fair comparison of SGAN and Bad-GAN, both models are trained on the same datasets, the sizes of which are listed in Table 4.1. For each input dataset (frequency-phase, time-phase, pulse-profile, and DM-curve) the labelled dataset consists of 25,828 training examples, 6,457 validation examples, and 13,837 testing examples as indicated in Table 4.1. The unlabelled dataset contains 20,000 examples. Besides, the same discriminator and generator architectures are also used for both models to prevent them from using a custom-tailored architecture that performs well on either of them (Li et al., 2019).

A regularisation technique known as label smoothing (Salimans et al., 2016) is used during the training to help the SGAN and Bad-GAN models to converge. Smoothing the label consists of replacing the label with a random value of around 0 for fake samples and around 1 for real samples. Real examples are labelled with a random value between 0.9 and 1.0, and fake examples are given a value between 0.0 and 0.2, for the DM-curve and pulse-profile datasets. In the case of frequency-phase and time-phase, these values are between 0.7 and 1.2 for real samples and between 0.0 and 0.3 for fake examples. These values were used by Balakrishnan et al. (2021) for SGAN and also performed well in Bad-GAN. In addition to label smoothing regularisation, the latter also suggested that occasionally flipping the labels during the training would improve the model performance. However, flipping the labels had no effect on the performance of Bad-GAN for each data. This technique is applied to the training of the frequency-phase and time-phase datasets for SGAN. For these two datasets, flipping the labels consists of replacing the fake example label with random values between 0.7 and 1.2, and the real example label with random values between 0.0 and 0.2. To investigate the contribution of the unlabelled data to the performance of SGAN and Bad-GAN, we train the models on different combinations of labelled and unlabelled data: 100, 500, 1000, 5000, 10000, and 20000 labelled data; each of these data is trained with 1000, 5000, 10000, and 20000 unlabelled examples. The performance of the models varies depending on the selected training examples. Therefore, each combination is trained five times with randomly selected examples and the median value is reported. For example, a combination of 100 labelled and 1000 unlabelled is trained five times, where each 100 and 1000 examples are all randomly selected. In this sense, the examples selected from these five selections can be completely independent when the data size is small; for larger data sizes, such as 10,000 examples, there is a higher probability of overlap between the different training sets. The implementation of SGAN and Bad-GAN (with the optimised weight parameters given in Section 4.5.1) can be summarised as follows:

1. Define the discriminator and the generator model architectures as described in Section 4.4.

2. Select randomly a number of labelled and unlabelled examples from the training set as mentioned above. For instance, 100 labelled examples and 1000 unlabelled examples.

3. Define the training algorithm for SGAN/Bad-GAN as discussed in Section 4.5.3 and 4.5.4.

4. This training algorithm is looped for 400 epochs using the selected labelled and unlabelled examples in Step 2. During these epochs, if the validation loss of the model is better than its validation loss in the previous epochs, then the model is saved as the best model.

5. Load the best model and evaluate it on the testing dataset using the evaluation metrics described in Section 2.4.5

6. Repeat Steps 2-5 five times. This is a compromise between checking how the performance varied depending on the exact training set used and the amount of computational time that it took to run each training. Thus, Five scores are obtained from each evaluation metric and the median value of each metric is reported in Section 4.6

The steps above are applied to all of the datasets, and the SGAN and Bad-GAN model performances on each dataset are discussed in the following sections.

4.6 Results and analysis

In this section, the results of classification scores are reported for each dataset: DM-curve, frequency-phase, pulse profile, and time-phase. Model training is performed with different labelled data sizes, from 100 to 20,000. Labelled data sizes from 100 to 5000 will be referred to as small labelled data sizes, and labelled data sizes from 10,000 to 20,000 will be referred to as large labelled data sizes. Besides comparing the performance of SGAN and Bad-GAN, we also perform two classification analyses: first, to visualise the importance of labelled data size on the performance of SGAN and Bad-GAN, the performance scores are reported for a fixed unlabelled size of 20,000 examples for all metrics for each dataset. Second, the classification scores for different numbers of unlabelled examples are reported to see whether increasing the size of unlabelled examples improves the model performance. Furthermore, for each dataset, the results are shown in bar plots and tables, where the bars correspond to the median values of the five iterations described in Step 6 above and the vertical black lines indicate the standard deviations or error bars. We compute all of the evaluation metrics as described above, however as these leads to an extremely large and often difficult to interpret data set we concentrate on using F1-score and Specificity metrics here as we did in Chapter 3 (the results with the other metric scores can be found in Appendix B). These two metrics are chosen as they provide a balancing score between the recall and precision scores (F1-score), and the proportion of true negative examples recovered by a model.

4.6.1 DM-Curve

This section presents a performance comparison between SGAN and Bad-GAN for the DM-curve dataset. The classification scores on the test set are highlighted in Table 4.4 for the performance of the models as a function of the labelled data size. Table 4.5 represents the results from training the models with different sizes of unlabelled examples, each with 1000 labelled examples. The results can also be visualised in Figure 4.6 and 4.7.

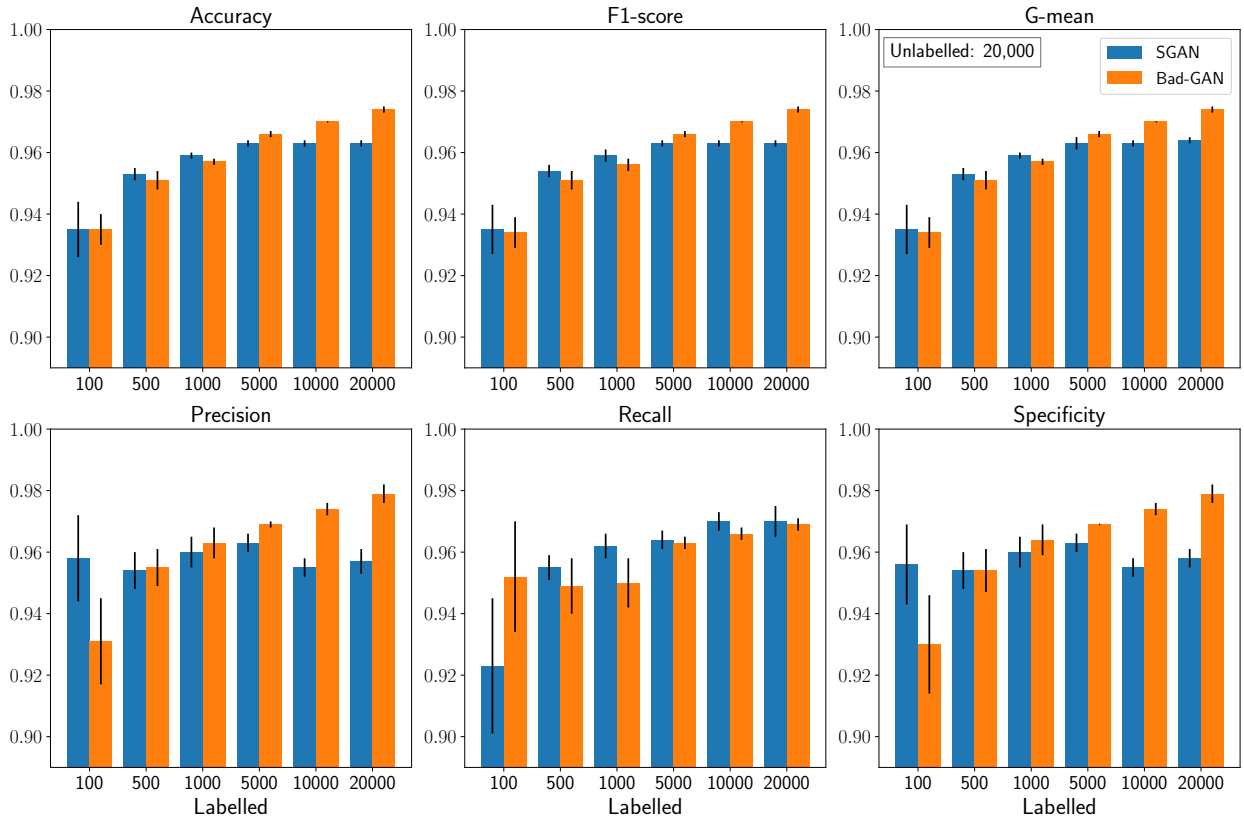

Figure 4.6: Performance on the DM-curve dataset: metrics to test the performance of SGAN and Bad-GAN using 20,000 unlabelled DM-curve examples. The x-axis corresponds to the number of labelled examples, while the y-axis represents each computed metric score. We note an increase in performance scores for the F1-score and specificity, as the number of labelled examples increases.

a) Model performance for increasing size of labelled DM-curve examples.

In Figure 4.6, we compare the performance of Bad-GAN (orange) and SGAN (blue) across a range of evaluation metrics described in Section 2.4.5. When comparing the two models, we can see that for small labelled data, the F1-score and specificity of SGAN and Bad-GAN are similar within the error bars. However, for a large labelled size, Bad-GAN outperforms SGAN: for 20,000 labelled examples, Bad-GAN achieved an F1-score of 0.974 and a specificity of 0.979, which are respectively 1.1% and 2.1% better than SGAN. As for the performance of the models with different sizes of the labelled dataset, we can see that Bad-GAN is more sensitive to the size of the labelled dataset than SGAN. For instance, from 1000 to 20,000

labelled examples, Bad-GAN F1-score increased by 1.8%, while SGAN F1-score increased by 0.04%. Overall, we note that with this configuration (Table 4.4), Bad-GAN outperforms SGAN.

Table 4.4: F1-score metric for SGAN and Bad-GAN models using 20,000 unlabelled DM-curve dataset.

Labelled	100	500	1000	5000	10000	20000
SGAN	0.935 ± 0.008	0.954 ± 0.002	0.959 ± 0.002	0.963 ± 0.001	0.963 ± 0.001	0.963 ± 0.001
Bad-GAN	0.934 ± 0.005	0.951 ± 0.003	0.956 ± 0.002	0.966 ± 0.001	0.970 ± 0.000	0.974 ± 0.001

b) Model performance for increasing size of unlabelled DM-curve examples.

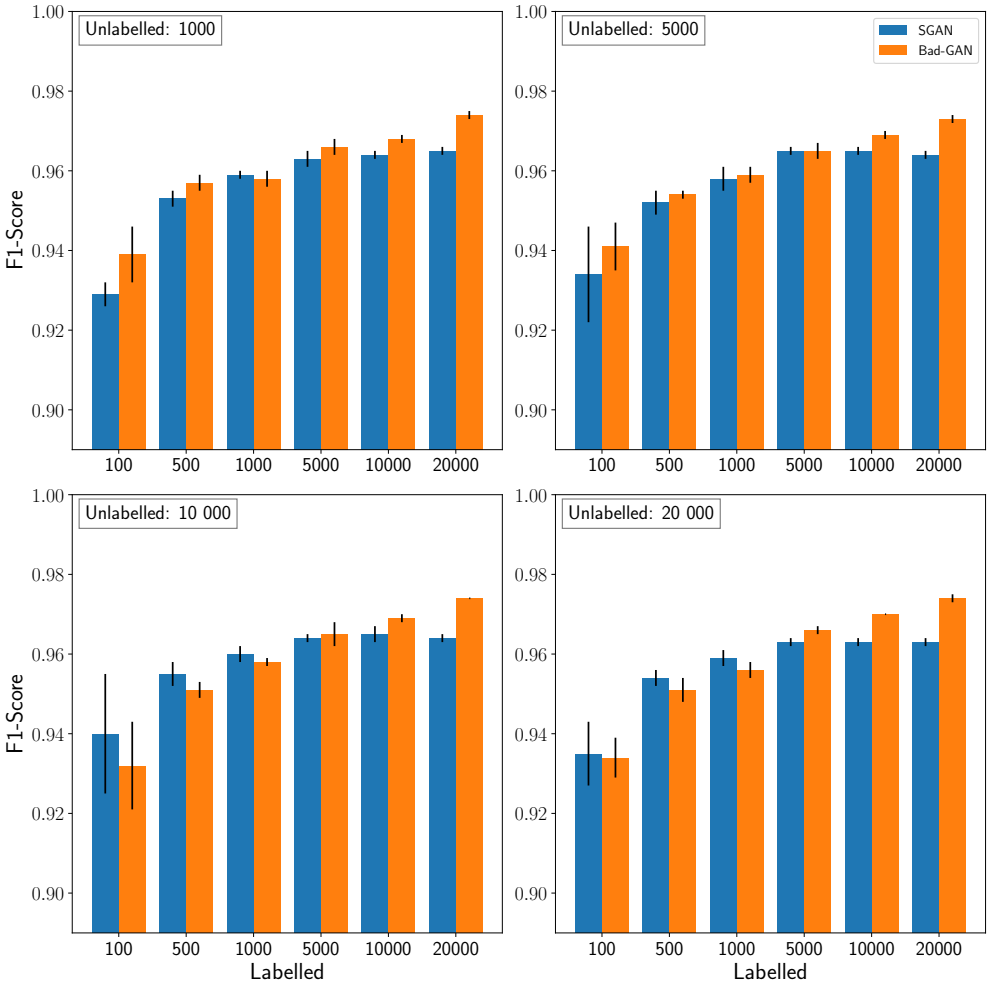

Figure 4.7: SGAN vs Bad-GAN performance for the DM-curve dataset with 1000 labelled examples using the F1-score metric. Increasing the number of unlabelled examples brings no remarkable improvements for SGAN and Bad-GAN, except the error bar for Bad-GAN is reduced in a few cases.

In this section, we consider the F1-score performance of SGAN and Bad-GAN for 1000 labelled data as a function of the size of unlabelled examples. Table 4.5 and Figure 4.7

show that the increasing amount of unlabelled data brings no significant improvement to the performance of either model.

Table 4.5: F1-score metric for SGAN and Bad-GAN models using 1000 labelled DM-curve dataset.

Unlabelled	1000	5000	10000	20000
SGAN	0.959 ±0.001	0.958 ±0.003	0.960 ±0.002	0.959 ±0.002
Bad-GAN	0.958 ±0.002	0.959 ±0.002	0.958 ±0.001	0.956 ± 0.002

4.6.2 Frequency-Phase

This section presents a performance comparison between SGAN and Bad-GAN for the frequency-phase dataset. The classification scores on the test set are highlighted in Table 4.6 for the performance of the models as a function of the labelled data size. Table 4.7 presents the results from training the models with different sizes of unlabelled examples, each with 1000 labelled examples. The results can also be visualised in Figure 4.8 and 4.9.

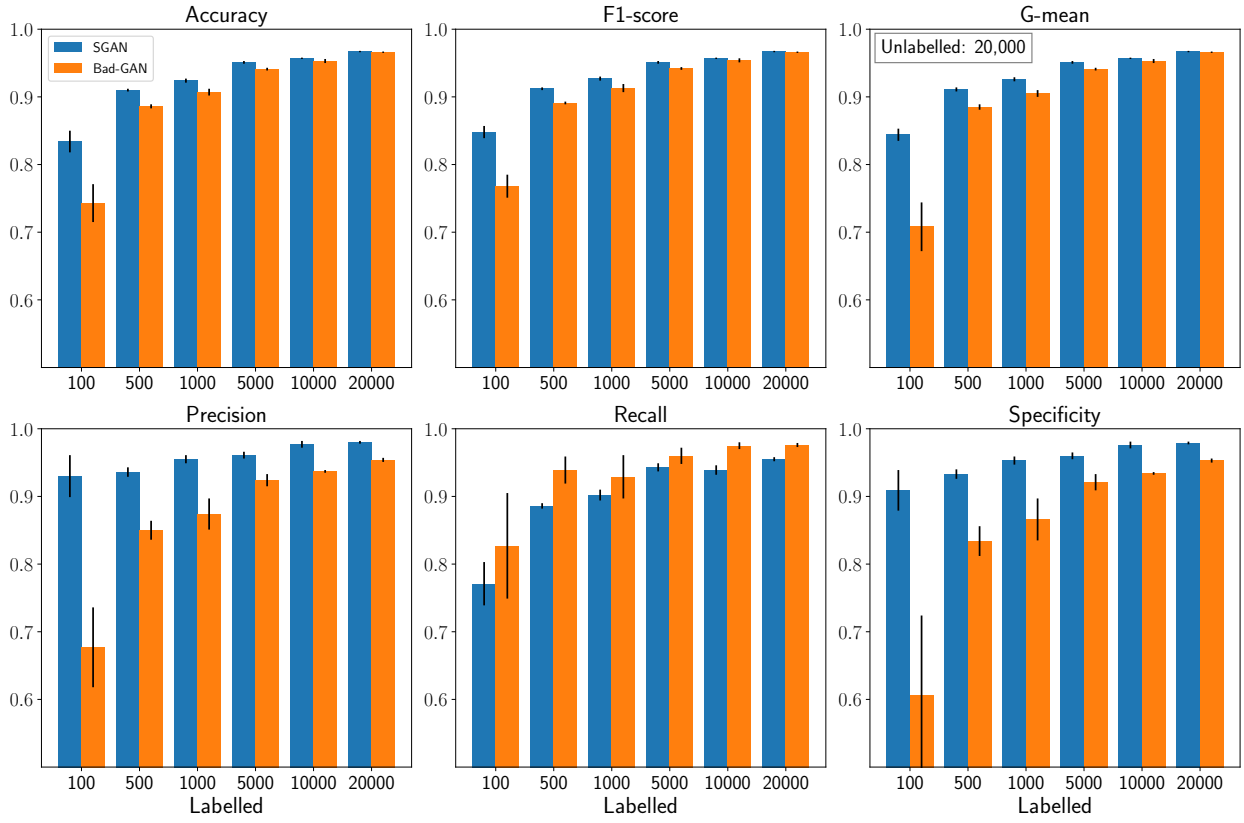

Figure 4.8: Various metrics to test the performance of SGAN and Bad-GAN using 20,000 unlabelled frequency-phase examples. The x-axis corresponds to the number of labelled examples, while the y-axis represents each computed metric score. We note an increase in performance scores for the F1-score and specificity, as the number of labelled examples increases.

a) Model performance for increasing size of labelled frequency-phase examples.

Regarding the frequency-phase dataset, Bad-GAN behaved differently compared to its performance on the DM-curve dataset. Overall, using the 20,000 unlabelled dataset, SGAN achieved better F1-score and specificity scores for all sizes of the labelled dataset, except for the F1-score where their performance is similar for labelled data above 5000. For instance, SGAN is 2.1% and 9.9% better than Bad-GAN in F1-score and specificity, respectively. On the other hand, the sensitivity of Bad-GAN to the size of labelled data is again observed in the frequency-phase dataset: for instance, from 1000 to 20,000 labelled data, Bad-GAN F1-score increased by 5.3%, while this increase is 4% for SGAN.

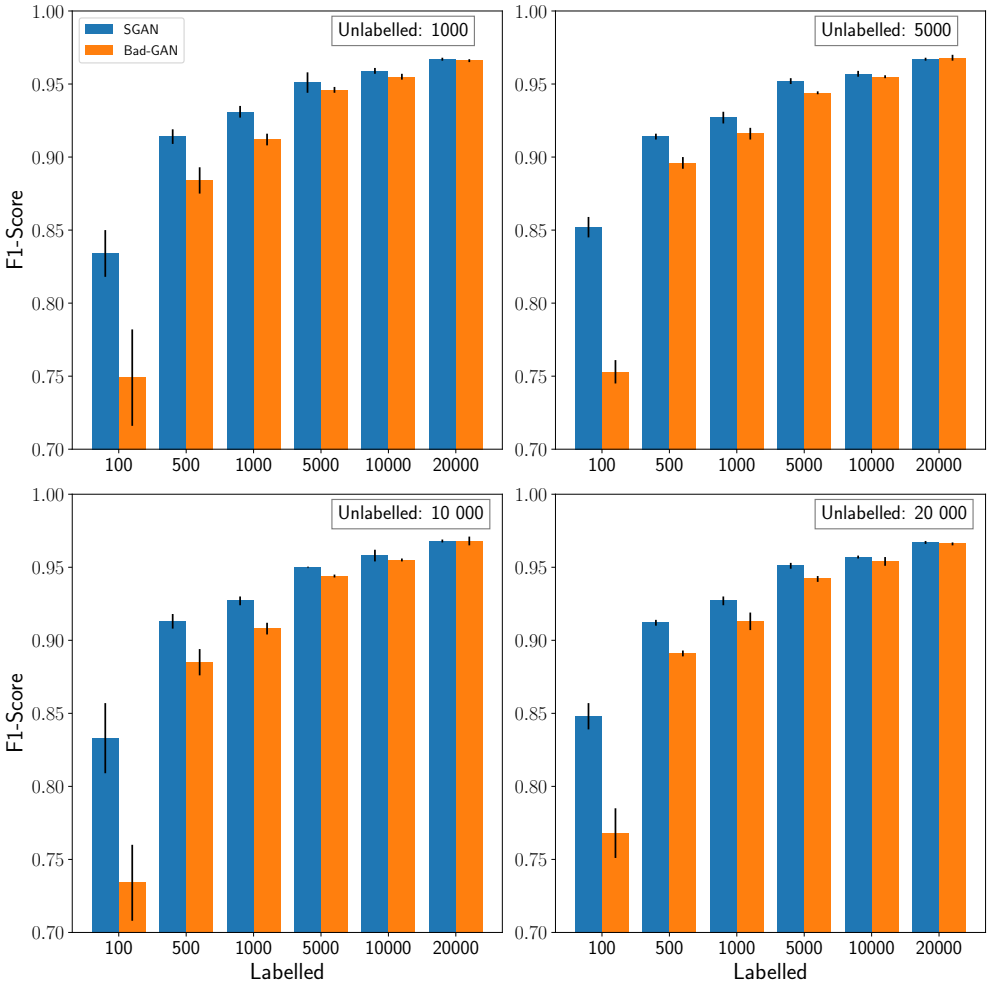

Figure 4.9: SGAN vs Bad-GAN performance for the frequency-phase dataset with 1000 labelled examples using the F1-score metric. Increasing the number of unlabelled examples brings no remarkable improvements for SGAN and Bad-GAN.

b) Model performance for increasing size of unlabelled frequency-phase examples.

Following the same procedures as in the DM-curve dataset, Table 4.7 shows that increasing the unlabelled size has no significant effect on the performance of SGAN and Bad-GAN regarding the F1-score.

Table 4.6: F1-score metric for SGAN and Bad-GAN models using 20,000 unlabelled frequency-phase examples.

Labelled	100	500	1000	5000	10000	20000
SGAN	0.848 ± 0.009	0.912 ± 0.002	0.927 ± 0.003	0.951 ± 0.002	0.957 ± 0.001	0.967 ± 0.001
Bad-GAN	0.768 ± 0.017	0.891 ± 0.002	0.913 ± 0.006	0.942 ± 0.002	0.954 ± 0.003	0.966 ± 0.001

Table 4.7: F1-score metric for SGAN and Bad-GAN models using 1000 labelled frequency-phase examples.

Unlabelled	1000	5000	10000	20000
SGAN	0.931 ± 0.004	0.927 ±0.004	0.927 ± 0.003	0.927 ± 0.003
Bad-GAN	0.912 ± 0.004	0.916 ± 0.004	0.908 ± 0.004	0.913 ± 0.006

4.6.3 Pulse-Profile

The performance comparison between SGAN and Bad-GAN for the time-phase dataset is presented here in a similar way as for the previous datasets. We compare the performance of Bad-GAN and SGAN using the pulse-profile dataset for various sizes of labelled examples and 20,000 unlabelled examples. The second part of the classification reports consists of the performance comparison of both models with increasing size of unlabelled examples for a fixed size of 1000 labelled examples.

a) Model performance for increasing size of labelled pulse-profile examples.

The performance of SGAN and Bad-GAN on the pulse-profile dataset is similar to what has been seen in the DM-curve dataset. In Table 4.8 and Figure 4.10, we can see that, from 500 to 5000 labelled examples, both models perform similarly with regards to F1-score and specificity. While for 100 labelled examples, SGAN performed better in F1-score than Bad-GAN, with an F1-score 8.8% better than Bad-GAN. However, for 10,000 and 20,000 labelled examples, Bad-GAN achieved higher scores in both F1-score and specificity than SGAN. For instance, for 20,000 labelled examples, the specificity of Bad-GAN is 6.7% better than SGAN. The sensitivity of Bad-GAN is also observed using the pulse-profile dataset: for instance, from 1000 to 20,000 labelled examples, Bad-GAN F1-score increased by 10.5%, while SGAN improved by 3.8%.

Table 4.8: F1-score metric for SGAN and Bad-GAN models using 20,000 unlabelled pulse-profile examples.

Labelled	100	500	1000	5000	10000	20000
SGAN	0.689 ± 0.014	0.732 ± 0.014	0.762 ± 0.014	0.796 ± 0.006	0.802 ± 0.008	0.8 ± 0.003
Bad-GAN	0.601 ± 0.074	0.733 ± 0.008	0.745 ± 0.009	0.785 ± 0.006	0.819 ± 0.004	0.85 ± 0.003

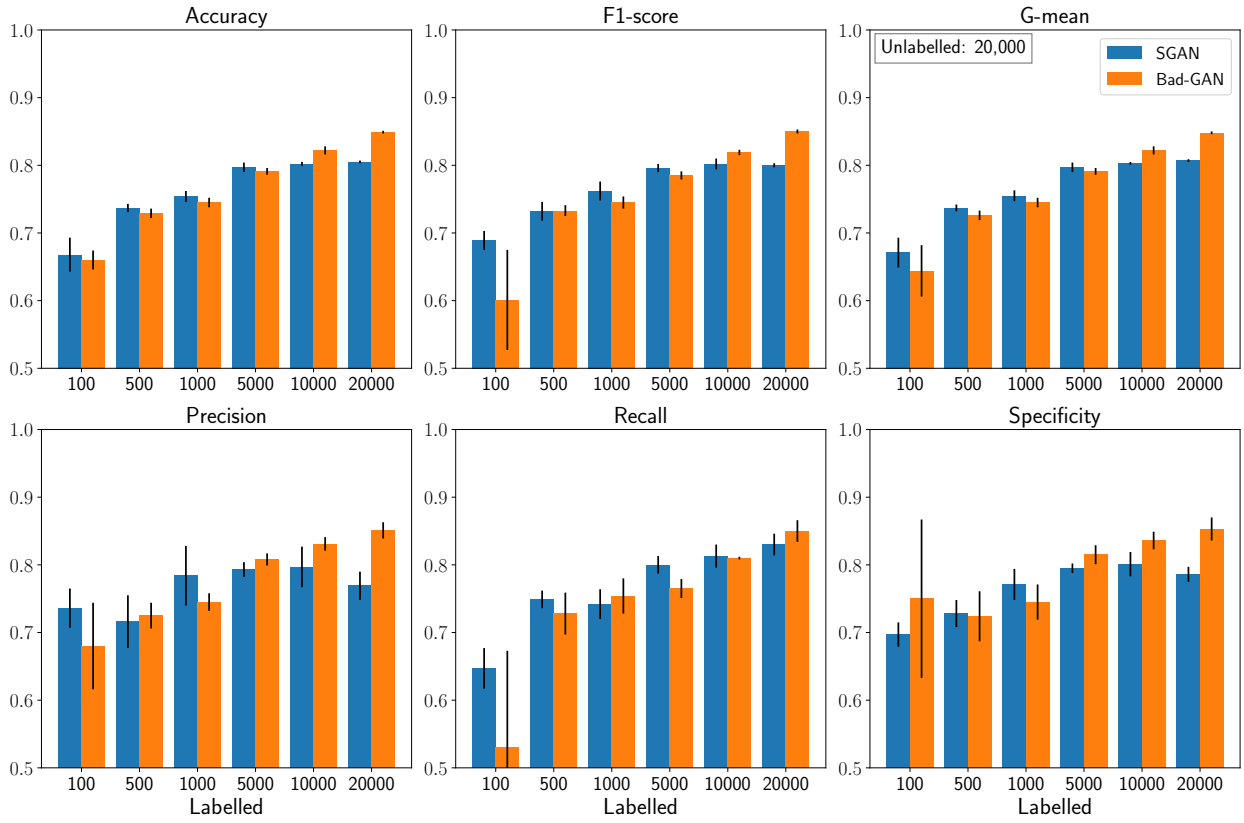

Figure 4.10: Various metrics to test the performance of SGAN and Bad-GAN using 20,000 unlabelled pulse-profile examples. The x-axis corresponds to the number of labelled examples, while the y-axis represents each computed metric score. We note an increase in performance scores for the F1-score and specificity, as the number of labelled examples increases, except for SGAN, which stops improving after 10,000 labelled examples.

b) Model performance for increasing size of unlabelled pulse-profile examples.

Here, we traine SGAN and Bad-GAN on different sizes of unlabeled examples as in the previous sections. Table 4.9 shows that increasing the number of unlabelled examples slightly improved the performance of both models on F1-score using 1000 labelled examples. In addition, SGAN performed slightly better than Bad-GAN for this configuration, with an F1-score 1.7% higher than that of Bad-GAN.

Table 4.9: F1-score metric for SGAN and Bad-GAN models using 1000 labelled pulse-profile examples.

Unlabelled	1000	5000	10000	20000
SGAN	0.755 ± 0.007	0.757 ±0.006	0.760 ± 0.003	0.762 ± 0.014
Bad-GAN	0.747 ± 0.005	0.748 ± 0.007	0.750 ± 0.006	0.745 ± 0.009

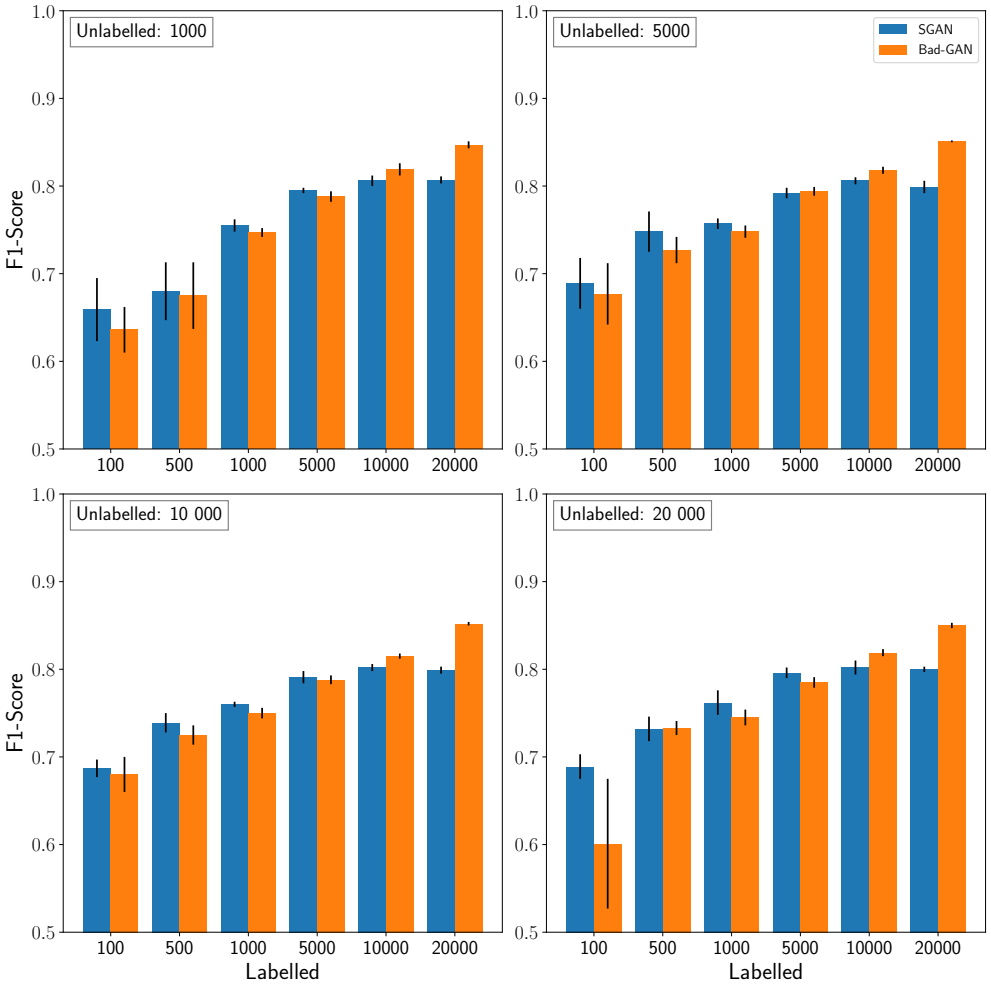

Figure 4.11: SGAN vs Bad-GAN performance for the pulse-profile dataset with 1000 labelled examples using the F1-score metric. Increasing the number of unlabelled examples brings no remarkable improvements for SGAN and Bad-GAN.

4.6.4 Time-Phase

This section presents the performance of SGAN and Bad-GAN using the time-phase dataset. The same steps are followed as for the other three datasets when comparing both models. First, we report the performance of SGAN and Bad-GAN for a fixed size of 20,000 unlabelled examples with various sizes of labelled examples. Second, the labelled example size is fixed at 1000 to investigate the performance of both models with varying numbers of unlabelled examples.

a) Model performance for increasing size of labelled time-phase examples.

The performance of SGAN and Bad-GAN for 20,000 unlabelled examples is shown in Table 4.10 and Figure 4.12. These results reveal that SGAN outperforms Bad-GAN in both F1-score and specificity across all sizes of the labelled dataset. However, the more the number of labelled examples increases, the more Bad-GAN performance comes closer to that of SGAN. For instance, Bad-GAN achieved a similar F1-score to SGAN for 20,000 labelled examples.

Again, the sensitivity of Bad-GAN can be seen for the time-phase dataset. From 1000 to 20,000 labelled examples, the F1-score of Bad-GAN improved by 12%, while SGAN F1-score increased by 10%.

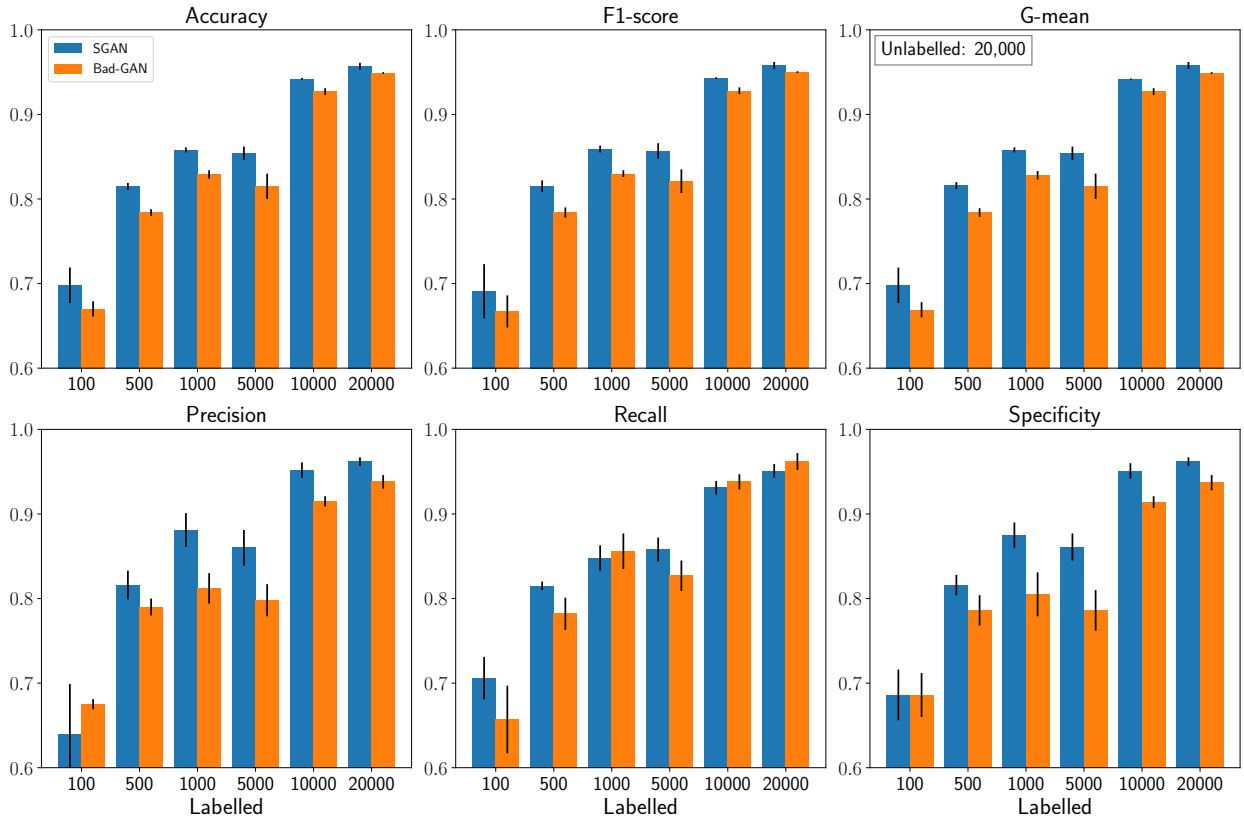

Figure 4.12: Various metrics to test the performance of SGAN and Bad-GAN using 20,000 unlabelled time-phase examples. The x-axis corresponds to the number of labelled examples, while the y-axis represents each computed metric score. We note an increase in performance scores for the F1-score and specificity, as the number of labelled examples increases.

Table 4.10: F1-score metric for SGAN and Bad-GAN models using 20,000 unlabelled time-phase examples.

Labelled	100	500	1000	5000	10000	20000
SGAN	0.691 ± 0.032	0.815 ± 0.007	0.859 ± 0.004	0.857 ± 0.009	0.943 ± 0.001	0.958 ± 0.004
Bad-GAN	0.667 ± 0.019	0.784 ± 0.006	0.830 ± 0.004	0.821 ± 0.014	0.928 ± 0.004	0.950 ± 0.001

b) Model performance for increasing size of unlabelled time-phase examples.

Here, we present the performance of SGAN and Bad-GAN for different sizes of unlabelled examples with a fixed size of 1000 labelled examples to analyse the contribution of the size of unlabelled examples to the performance of both models. Interestingly, Table 4.11 and Figure 4.13 show that increasing the number of unlabelled data from 1000 to 20,000 improved the F1-score performance of Bad-GAN by 2.1%, while it is only 0.05% for SGAN. This is the highest improvement for Bad-GAN across all the other datasets. Interestingly, while increasing the

number of unlabelled examples has no significant improvement for SGAN trained on each dataset.

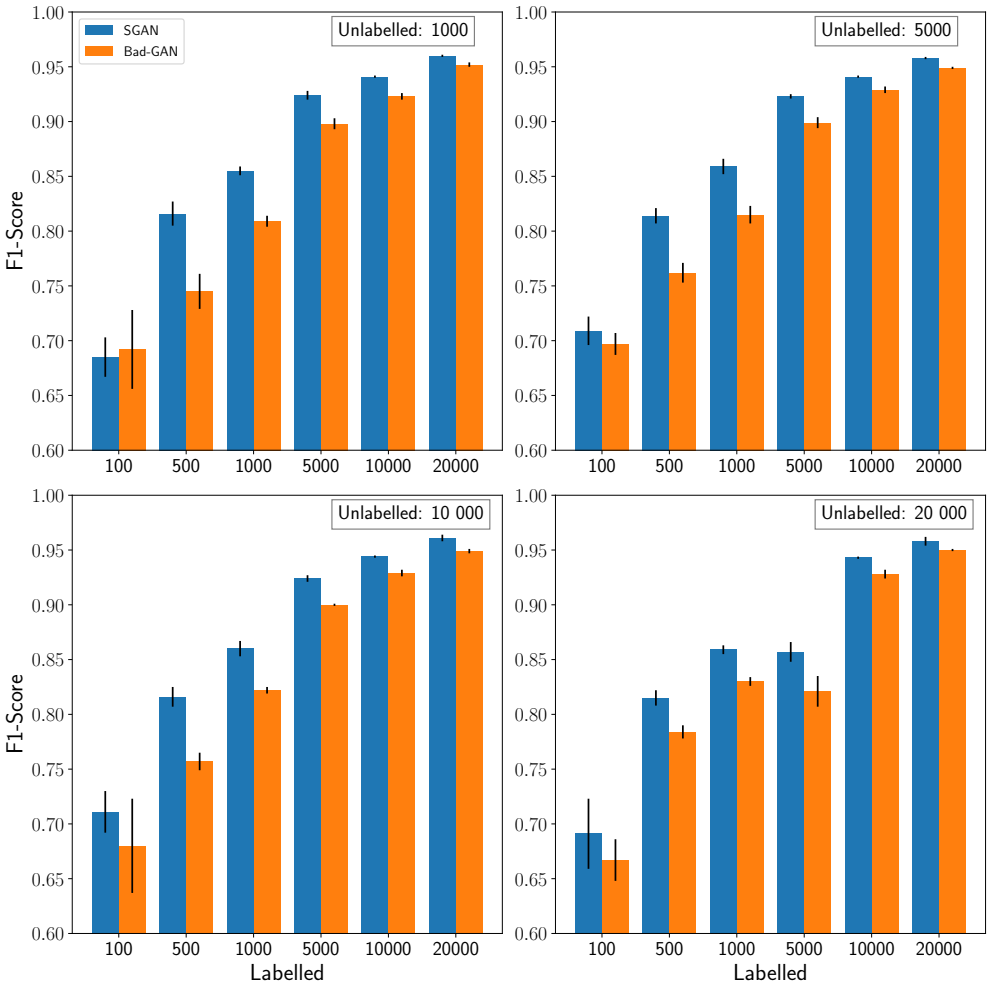

Figure 4.13: SGAN vs Bad-GAN performance for the time-phase dataset with 1000 labelled examples using the F1-score metric. Increasing the number of unlabelled examples brings no remarkable improvements for SGAN and Bad-GAN, except in Bad-GAN for 500 and 1000 labelled examples.

Table 4.11: F1-score metric for SGAN and Bad-GAN models using 1000 labelled time-phase examples.

Unlabelled	1000	5000	10000	20000
SGAN	0.855 ± 0.004	0.859 ±0.007	0.860 ± 0.007	0.859 ± 0.004
Bad-GAN	0.809 ± 0.005	0.815 ± 0.008	0.822 ± 0.003	0.830 ± 0.004

4.6.5 Overall performance of SGAN and Bad-GAN on each dataset

SGAN and Bad-GAN exhibit various performance scores for each dataset: one model performs better than another depending on the type and size of the dataset. The performance

of both models is greatly dependent on the size of labelled examples, especially for Bad-GAN which is more sensitive to the number of labelled examples compared to SGAN. On the other hand, increasing the number of unlabelled examples had no substantial improvement to the performance of both models regarding the F1-score metric, except for Bad-GAN trained on the time-phase dataset. In some cases, using more unlabelled examples worsened the performance of both models. For instance, it can be seen when the models are trained on 1000 labelled examples with 10000 or 20000 unlabelled examples. Choosing the number of unlabelled examples to be trained with a number of labelled examples is therefore crucial to fully harness the importance of unlabelled examples in improving the model performance. Furthermore, using the smallest number of labelled examples (100), the highest scores Bad-GAN achieved are an F1-score of 0.941 and a specificity of 0.942 using 1000 unlabelled examples in the DM-curve dataset. SGAN, on the other hand, produced the highest scores using 10,000 unlabelled examples in the DM-curve dataset, with an F1-score of 0.940 and a specificity of 0.952. For all of the experiments in this work, the highest performance achieved by Bad-GAN is an F1-score of 0.974 and a specificity of 0.979 using 20,000 labelled and 20,000 unlabelled examples in the DM-curve dataset. The highest performance achieved by SGAN is an F1-score of 0.968 and a specificity of 0.981 using 20,000 labelled and 10,000 unlabelled examples in the frequency-phase dataset. The following section aims to improve the performance of SGAN and Bad-GAN by combining the performance of each model trained on each dataset.

4.6.6 Combined model performance

In this section, the combination method of the four models from DM-curve, frequency-phase, pulse-profile, and time-phase is described. In the previous sections, each of these models is trained five times on the training set. As a result, there are five instances of each of the four models saved during the training. The experiments consist of stacking the performance of each of the 5-fold of the four models. In other words, one fold of the model contains four models. At the end of the stacking procedures, five different classification performances are obtained and the median values of the metric scores are reported. Besides, five pairs of labelled and unlabelled training data are used: (100, 1000), (1000, 5000), (5000, 10000), (10000, 20000), and (20000, 20000). Therefore, the models saved on these settings are used in the model stacking. To obtain the stacked performance, the steps below are performed for each fold of four models:

1. The first step consists of loading the four models trained on the above training data. For example, a model trained on 100 labelled and 1000 unlabelled examples.

2. Select a number of labelled data randomly from the training set (DM-curve, frequency-phase, pulse-profile, and time-phase). As in the example above, 100 labelled examples are selected randomly from each training set respectively. Note that the four training sets have the same targets.

3. The next step is to make a prediction of these examples using the loaded models. The DM-curve loaded model is used to predict the 100 DM-curve examples, the frequency-phase model made a prediction on the 100 frequency-phase examples, and so forth.

4. The results of the predictions from the four models are stacked. Using a similar example as above, the stacked result is 100×4 tabular data.

5. Logistic regression is used, as in Balakrishnan et al. (2021), to train this tabular data to obtain the stacked model, which is the trained logistic regression model. Note that we have tested ANN and SVM classifiers for combining the performance of each model. However, the results are similar to those obtained using logistic regression.

6. To test the performance of the stacked model, the same testing set used to evaluate the individual model is loaded for each of the four training sets. Then, the four loaded models are used to predict these testing sets as in Step 3. As a result, tabular data is obtained as in Step 4. The logistic model (stacked model) is evaluated using these tabular data.

The steps above are followed to obtain the stacked model for SGAN and Bad-GAN. The resulting performance of SGAN and Bad-GAN is represented in Figure 4.14. This figure shows the performance of SGAN/Bad-GAN models trained on different pairs of labelled and unlabelled (labelled-size, unlabelled-size) examples: (100, 1000); (1000, 5000); (10000, 20000); and (20000, 20000). The performance of the models on each dataset is also presented along with the combined performance: pulse-profile (PP), time-phase (TP), frequency-phase (FP), and dm-curve (DM). As before, the F1-score and specificity scores are used to compare the performance of SGAN and Bad-GAN. However, the full results using other metric scores can be found in Appendix C. The grey dashed lines in this figure illustrate the performance of both models on the smallest and largest number of labelled examples.

These results reveal that for 20,000 labelled examples, SGAN performed slightly better than Bad-GAN, with a difference in F1-score and specificity of 0.6% and 0.4% respectively. On the other hand, for 100 labelled examples, Bad-GAN outperformed SGAN, especially for the specificity score: Bad-GAN achieved an F1-score of 0.7% and a specificity of 4.4% better than SGAN. The difference in specificity score using a small number of labelled examples is remarkable in the comparison of SGAN and Bad-GAN. This means that for 6918 negative examples in the testing set, Bad-GAN recovered 6541 of them, while SGAN is able to recover 6247 of the negative examples.

Comparing the individual performance of SGAN and Bad-GAN with the combined model, Figure 4.14 shows that Bad-GAN F1-score and specificity scores on DM-curve are similar to those of the combined model for 100 and 1000 labelled examples. Using the DM-curve dataset could be sufficient to achieve the performance of the combined model for these numbers of labelled examples. SGAN, on the other hand, achieved the same F1-score as the combined model using 100 DM-curve labelled examples, while its performance on the specificity is

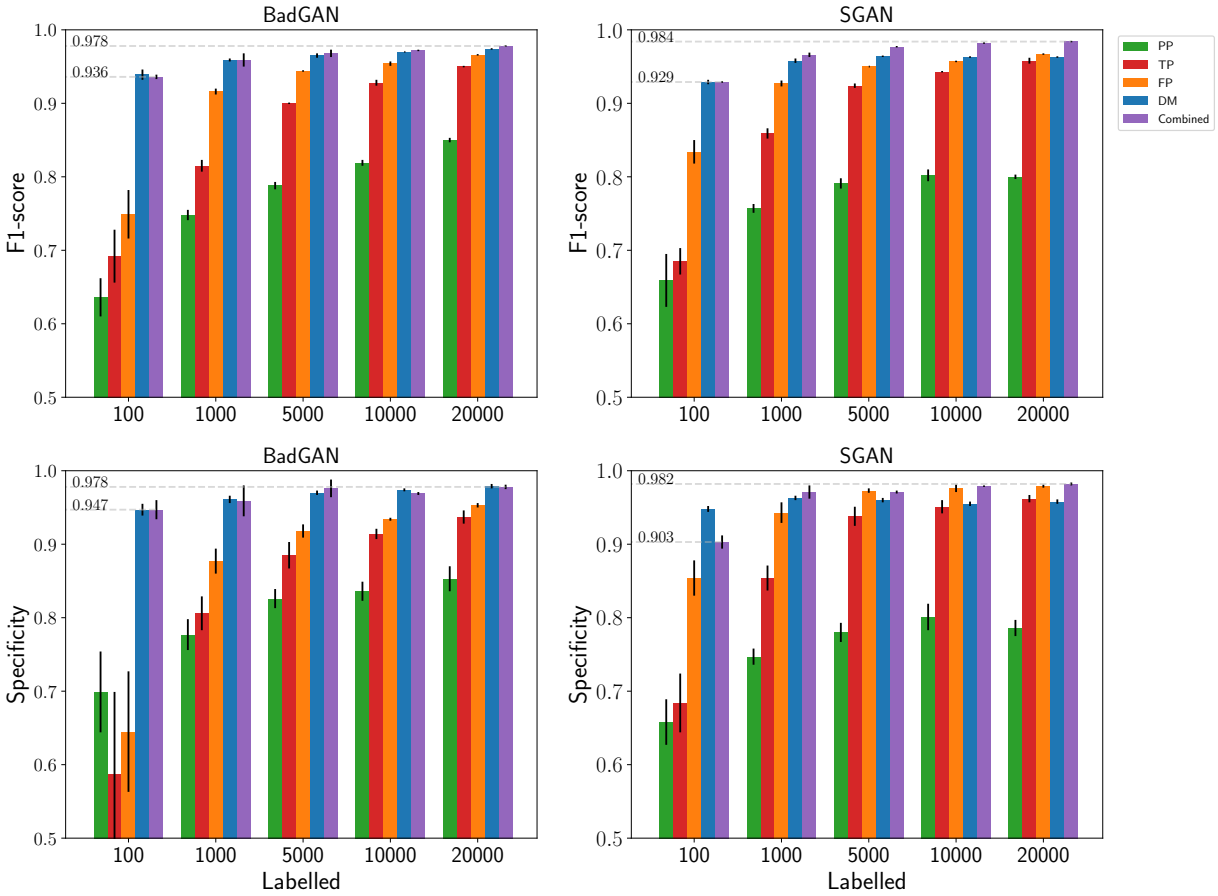

Figure 4.14: Combined performance of SGAN and Bad-GAN models: the performance of these models trained on pulse-profile (PP, green), time-phase (TP, red), frequency-phase (FP, orange), and DM-curve (DM, blue) is combined to obtain the combined performance (purple). Each bar represents the median value of five performance scores, while the black vertical lines on each bar represent standard deviations (errors) of the scores. Each labelled size, 100, 1000, 5000, 10000, and 20000 corresponds to a model training with 1000, 5000, 10000, 20000, and 20000 unlabelled examples respectively. The grey dashed horizontal lines represent the performance of the combined model on 100 and 20000 labelled examples.

better than that of the combined model for this labelled data size: training SGAN on the DM-curve only could be sufficient to achieve the combined performance for 100 labelled examples. Overall, no significant improvement is seen in Bad-GAN when the performance of each model is combined, while there is a moderate increase in F1-score for SGAN. This may be due to the fact that Bad-GAN performed poorly on the frequency-phase, time-phase, and pulse-profile, and that the performance of Bad-GAN between these three datasets itself is different. For SGAN, its performance on the DM-curve, time-phase, and frequency-phase is almost similar for a larger number of labelled examples, which might be the reason for the improvement in the combined model performance.

Overall, combining the performances of the SGAN and Bad-GAN models slightly in-

creased their performance compared to their highest performance when trained on the individual dataset: for Bad-GAN, its best F1-score increased only by 0.004%, while there is no improvement in the specificity. In the case of SGAN, its highest scores are improved by 0.016% for the F1-score and by 0.001% for the specificity.

4.7 Conclusion of the chapter

A study of the performance of SGAN and Bad-GAN is performed in this chapter, using four types of datasets: DM-curve, frequency-phase, pulse-profile, and time-phase. SGAN is reproduced from the works of Balakrishnan et al. (2021) who carried out a similar study. Bad-GAN is a semi-supervised approach that is intended to improve the stability of semi-supervised GANs by implementing different loss functions in the discriminator and generator models. The data is divided into three categories: training, validation, and testing sets. The training set contains labelled and unlabelled examples. After building and optimising the models, different data configurations were used to train SGAN and Bad-GAN: 100−20,000 labelled examples were trained with 1000−20,000 unlabelled examples for each dataset. Also, the performance of both models on each dataset is combined with the aim to observe improvements in the combined performance. F1-score and specificity metrics were used to discuss and compare the performance of SGAN and Bad-GAN. The main findings of this study are as follows:

- SGAN and Bad-GAN both achieved similar results in most cases, which is a function of the type of datasets and the number of labelled and unlabelled examples used during model training. For a fixed size of unlabelled examples, increasing the number of labelled examples greatly improved the performance of both models, where Bad-GAN is more sensitive to the size of labelled examples than SGAN. For a fixed number of labelled examples, increasing the number of unlabelled examples resulted in poor improvements in the performance of both models. The choice of the number of unlabelled examples to be trained with a specific size of labelled examples is also crucial as it can improve or worsen the performance of the models.

- Overall, Bad-GAN worked best on the DM-curve dataset, achieving an F1-score of 0.974 and a specificity of 0.979 using 20,000 labelled and 20,000 unlabelled examples. SGAN, on the other hand, produced its best performing score on the frequency-phase dataset, with an F1-score of 0.968 and a specificity of 0.981, using 20,000 labelled and 10,000 unlabelled examples.

- Combining the performance of the models from each dataset produced a small improvement in Bad-GAN and a moderate increase in F1-score for SGAN.

Conclusion & Future Work

Given the growing volume of pulsar data in pulsar surveys, visual inspection of these data is infeasible for human inspection. State-of-the-art techniques to boost pulsar classification have become of primary importance for pulsar searching. The main focus of this thesis was on the study of machine learning techniques for pulsar candidate classification to explore different approaches that could improve the performance of existing machine learning classifiers.

The first part of this thesis focused on pulsar candidate classification in the HTRU1 and HTRU2 datasets. HTRU1 contains 1196 real pulsars and 89,996 non-pulsars, while HTRU2 contains 1639 real pulsars and 16,259 non-pulsar examples. 40% and 20% of the entire dataset were used as testing sets for HTRU1 and HTRU2 respectively. We examined the performance of three machine learning classifiers on these datasets, including Random Forest, XGBoost, and ANNs. Before training the classifiers, we carried out feature selection using feature importance scores computed from Random Forest and XGBoost. We selected the eight most important features provided by these two classifiers. Then, we optimised Random Forest and XGBoost using Bayesian optimisation, while a 5-cross validation was used to optimise the ANNs model. The performance of each model was evaluated using F1-score and specificity. Since the two datasets are imbalanced, data sampling techniques were used to balance the number of positive and negative classes in the datasets: SMOTEENN was used for HTRU1 and ENN for HTRU2. Feature selection and model optimisation were again performed before training the classifiers on the re-sampled datasets, and the major findings of the work are as follows:

- The skewness of the integrated pulse profile is the most discriminative feature found by Random Forest and XGBoost for HTRU1 and HTRU2.
- Regarding the best performing classifiers on HTRU1, XGBoost achieved an F1-score of 0.970 and a specificity of about 1.000, while Random Forest produced an F1-score of 0.954 and a specificity of 1.000. As for HTRU2, XGBoost outperformed the other two classifiers with an F1-score of 0.919 and a specificity of 0.996.
- Re-sampling the datasets only brought improvements on the recall scores, while the F1-score and specificity scores dropped for each dataset.

Using data sampling techniques could improve the model performance in different evaluation metrics. However, if a specific metric score needs to be improved, a further study of

different parameters and testing are required, such as the number of k-nearest neighbours or the sampling strategy used in the sampling methods (e.g., 1:1 or 1:2 ratio). Besides, it is most probably that the HTRU2 dataset is a complex/noisy dataset, and therefore, harder to classify than the HTRU1 dataset. In addition, the nature of the HTRU2 dataset might be more directly related to a real-time classification environment that might be needed for future telescopes. Examining other feature selection approaches, which are independent from the algorithms, could improve the classification performance on HTRU2.

The second part of the work in this thesis consists of studying the performance of SGAN and Bad-GAN to explore the semi-supervised learning approach in a scenario where labelled datasets are scarce. Four image datasets from the HTRU-S Lowlat survey, processed by Balakrishnan et al. (2021), were used in this work. Each of these datasets contains 84,694 labelled pulsar candidates, where 20,000 of them were set apart as unlabelled datasets and the rest as labelled datasets, to work with the semi-supervised method. The labelled dataset was split into a training set (70%, including the validation set), and a testing set (30%). We trained the optimised SGAN and Bad-GAN models on different sizes of training examples, ranging from 100 to 20,000 labelled examples, and trained them with 1000 to 20,000 unlabelled examples. This was to examine the performance of each model as a function of the number of labelled and unlabelled data. That will help us to consider which model might be best for future pulsar surveys when they first start. Besides, the performances of the models trained from each dataset were combined to improve the performance of the models. Also, F1-score and specificity were used to evaluate the performance of the models. The major findings in this work include:

- SGAN and Bad-GAN performed similarly in most labelled and unlabelled data size configurations, and their performance was also dependent on the type of dataset used.
- Increasing the amount of unlabelled data brought no substantial improvement in the performance of either model. Balakrishnan et al. (2021) found similar results and concluded that training the SGAN with 50,814 labelled examples requires more than 200,000 unlabelled candidates to see an improvement in the model's performance.
- Using the smallest size of 100 labelled data, both models performed best on the DM-curve dataset. SGAN achieved an F1-score of 0.940 and a specificity of 0.952, while an F1-score of 0.941 and a specificity of 0.942 were obtained from Bad-GAN.
- The highest scores achieved by both models from all of the experiments were an F1-score of 0.968 and a specificity of 0.981 for SGAN using the frequency-phase dataset, and an F1-score of 0.974 and a specificity of 0.979 for Bad-GAN using the DM-curve dataset.
- Combining the performance of the models from each dataset produced no significant improvement in Bad-GAN and a moderate increase in F1-score for SGAN using more

than 10,000 labelled examples. This is consistent with the findings of Balakrishnan et al. (2021), where they started to find a substantial improvement in SGAN from 10,000 labelled examples.

- The highest combined performance scores achieved by both models were an F1-score of 0.978 and a specificity of 0.978 for Bad-GAN, and an F1-score of 0.984 and a specificity of 0.982 for SGAN.

It is worth noting that the Balakrishnan et al. (2021) work achieved combined performance with an F1-score of 0.989 and a specificity of 0.983 using 50,814 labelled training sets with 265,172 unlabeled examples, tested on 21,173 pulsar candidates.

We have seen that, while Bad-GAN performs almost identically to SGAN, the latter performs better with smaller labelled datasets and thus may be more appropriate when a pulsar survey first starts. However, as the survey progresses and more positive labelled data becomes available, one may consider switching to Bad-GAN.

The performance of Bad-GAN could be possibly improved by considering the following:

- It has been shown in other studies, non-related to pulsar classification, that Bad-GAN is sensitive to the value of batch size (Lecouat et al. 2018;Li et al. 2019). Thus, it is worth investigating the effect of varying the batch size on the overall performance of Bad-GAN.

- Work needs to be done in the implementation of the density estimation model for pulsar data to generate samples in low-density areas of the feature space. For instance, the PixelCNN+ model was used by Dai et al. (2017) as a density estimator when training Bad-GAN on natural image data. As discussed in this thesis, the latter could improve the performance of the Bad-GAN model.

- Finding a robust method to find the optimal loss weights, as presented in Table 4.2, would help to increase the performance of Bad-GAN.

- Different architectures of the discriminator and the generator models might change the performance of the Bad-GAN model.

Finally, it has been demonstrated (for example, by Tan et al. 2018) that transferring models trained on data from one telescope to another is not as effective as one would expect, even when the method is particularly designed to do so. Therefore, it may be required to develop a new classifier for each new telescope, increased observing frequency, or any other substantial changes. As a result, it affects how a pulsar survey is planned. Does one need to plan to observe a set of known sources with the parameters of the new survey to build a new classifier, first, and then proceed with the survey of new pulsars? If this is the case, then one wants to obtain a minimal sample to not use too much precious observing time. This might become even more relevant when considering how observing conditions might change throughout a

survey (which might take years). In particular, the RFI situation might mean more false positives, or less bandwidth available. Thus, continuing to investigate the best methods of dealing with small labelled data sets and high class imbalance is an important part of making the next generation of pulsar surveys as successful as possible for a given amount of observing time.

Appendix A

Software packages

A.1 SGAN

The SGAN model was run based on Python 3.8.5 environment using the Keras library (version 2.4.3), with a Tensorflow backend (version 2.5.0). The work[1] of Balakrishnan et al. (2021) was used to train SGAN using four Python classes for the four datasets: Train_SGAN_DM_Curve, Train_SGAN_Pulse_Profile, Train_SGAN_Freq_Phase, and Train_SGAN_Time_Phase.

A.2 Bad-GAN

The Bad-GAN used in this work was inspired by the Bad-GAN original paper[2] Dai et al. (2017). Their work was based on Python 2.7+ and Pytorch 0.1.12, while our work was based on Python 3.8.5 and Pytorch 1.7.1. Many changes were made to their code to make it compatible with the Python environment we used for this project, as they were using outdated Python packages. In addition, customised data loaders were created to load our datasets as they are required for the training of Bad-GAN in Pytorch. The implementation of Bad-GAN in this thesis can be found at GitHub.[3].

[1] `https://github.com/vishnubk/sgan`
[2] `https://github.com/kimiyoung/ssl_bad_gan`
[3] `https://github.com/rtprincy/MPhil-project`

Appendix B

Performance of SGAN and Bad-GAN on the individual dataset

B.1 Performance of SGAN and Bad-GAN across other metrics score using the DM-curve dataset

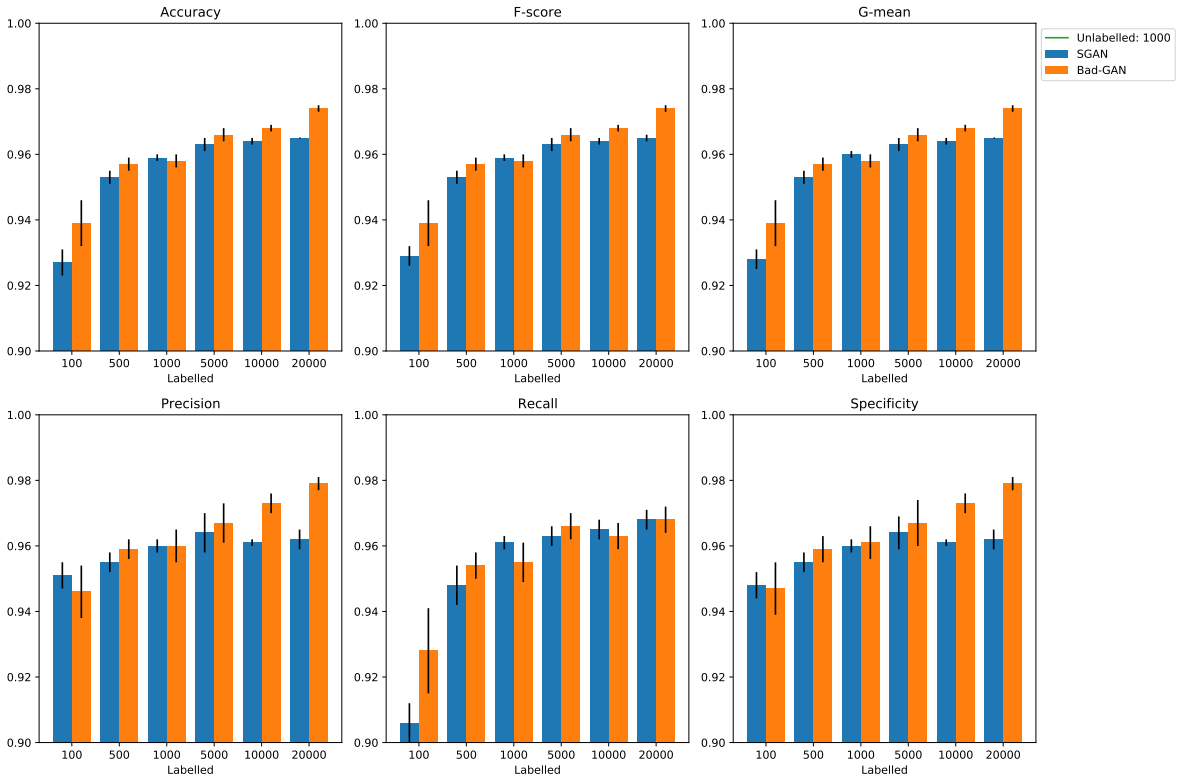

Figure B.1: Performance on the DM-curve dataset: metrics to test the performance of SGAN and Bad-GAN using 1000 unlabelled DM-curve examples.

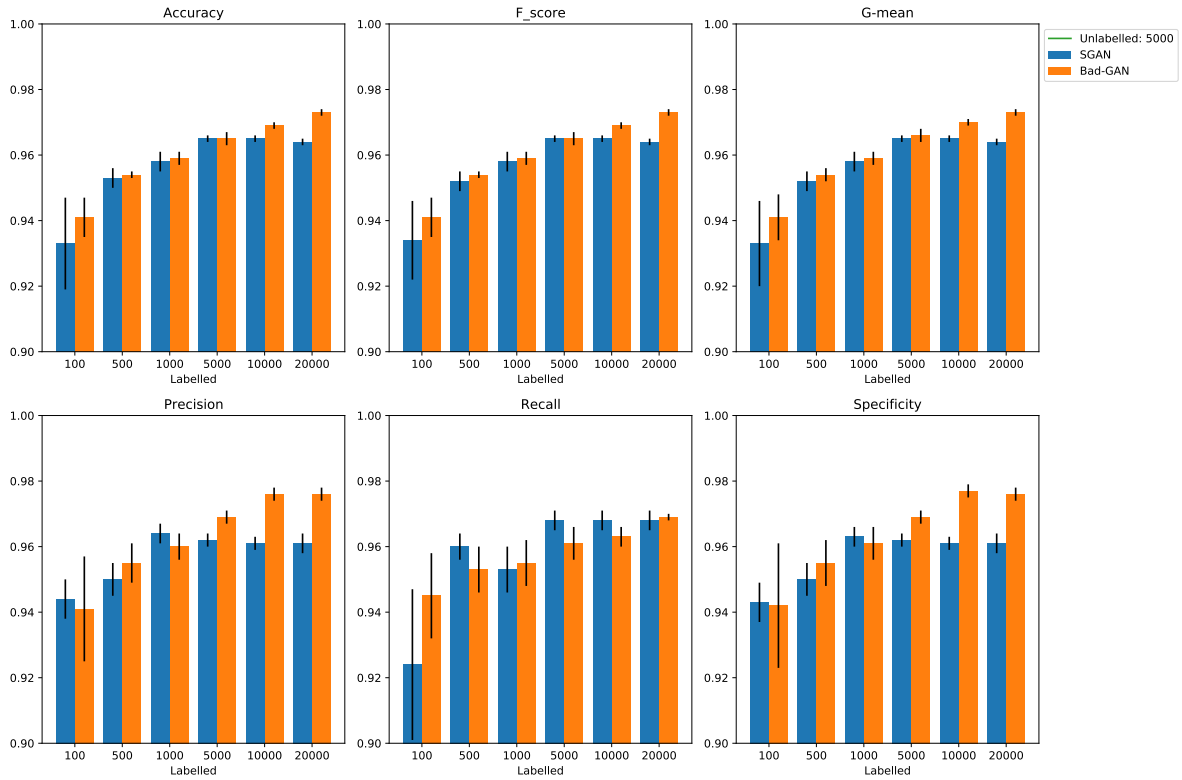

Figure B.2: Performance on the DM-curve dataset: metrics to test the performance of SGAN and Bad-GAN using 5000 unlabelled DM-curve examples.

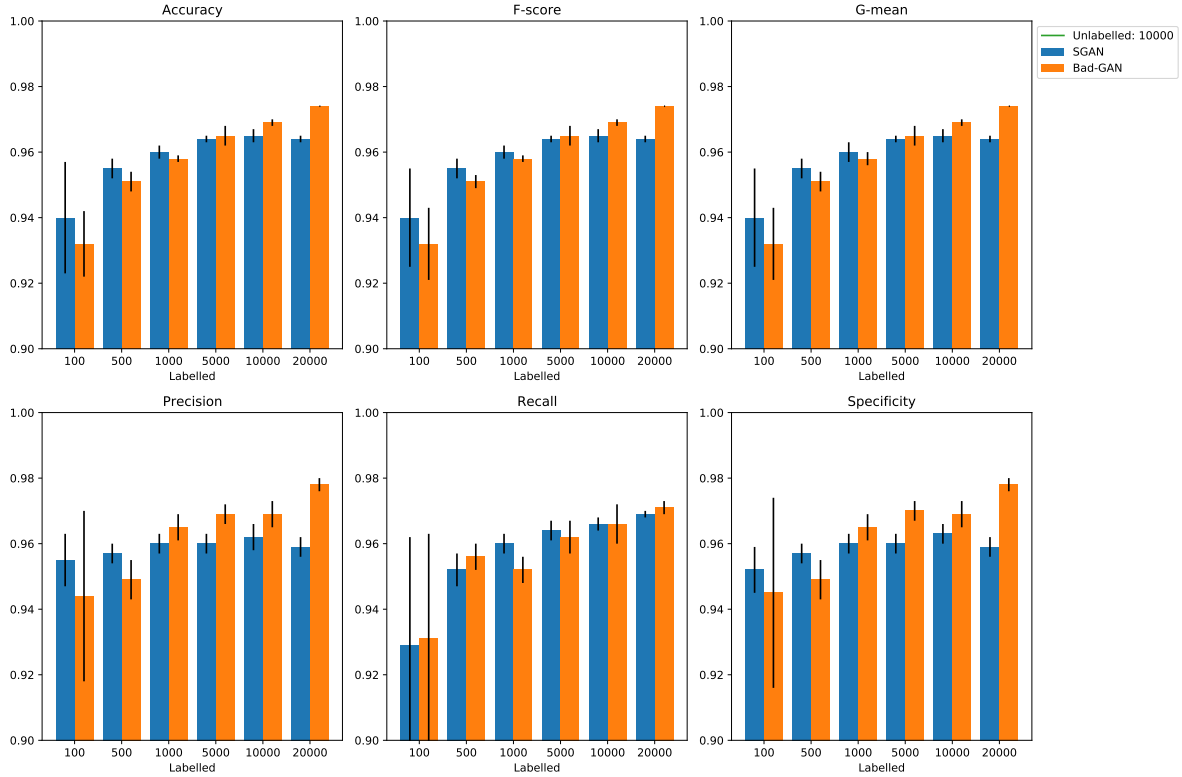

Figure B.3: Performance on the DM-curve dataset: metrics to test the performance of SGAN and Bad-GAN using 10,000 unlabelled DM-curve examples.

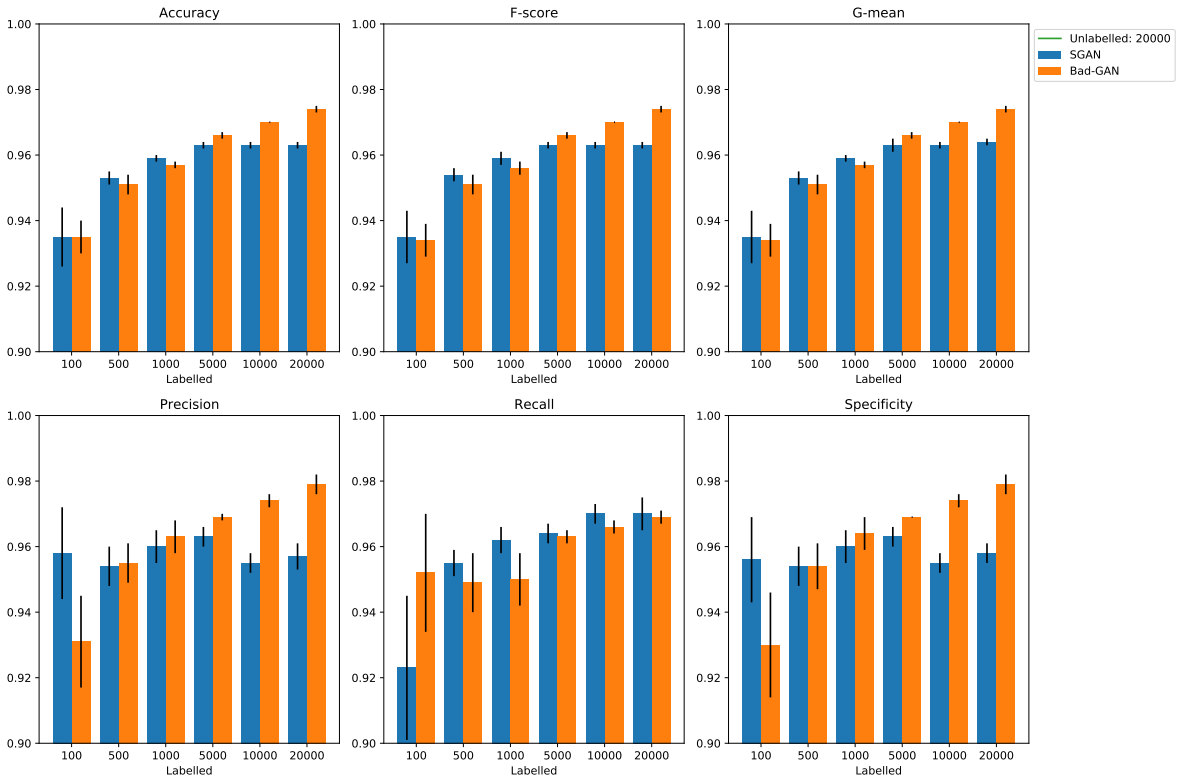

Figure B.4: Performance on the DM-curve dataset: metrics to test the performance of SGAN and Bad-GAN using 20,000 unlabelled DM-curve examples.

B.2 Performance of SGAN and Bad-GAN across other metrics score using the frequency-phase dataset

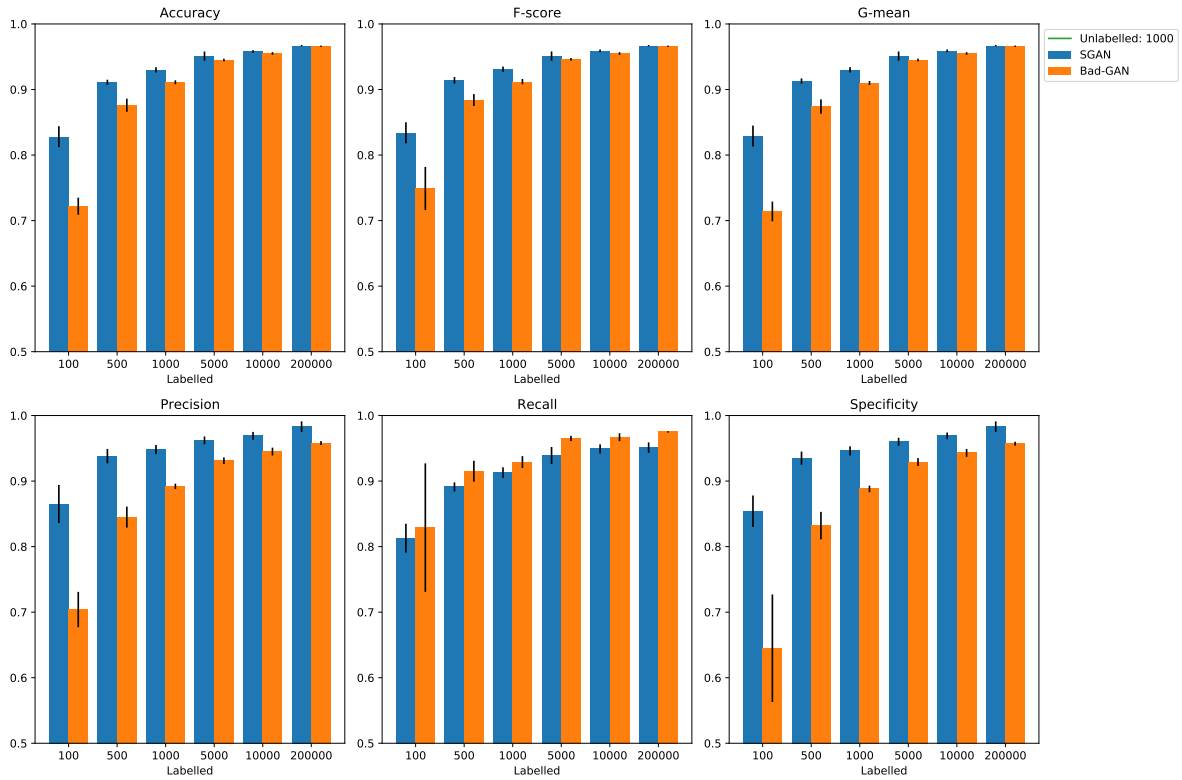

Figure B.5: Performance on the frequency-phase dataset: metrics to test the performance of SGAN and Bad-GAN using 1000 unlabelled frequency-phase examples.

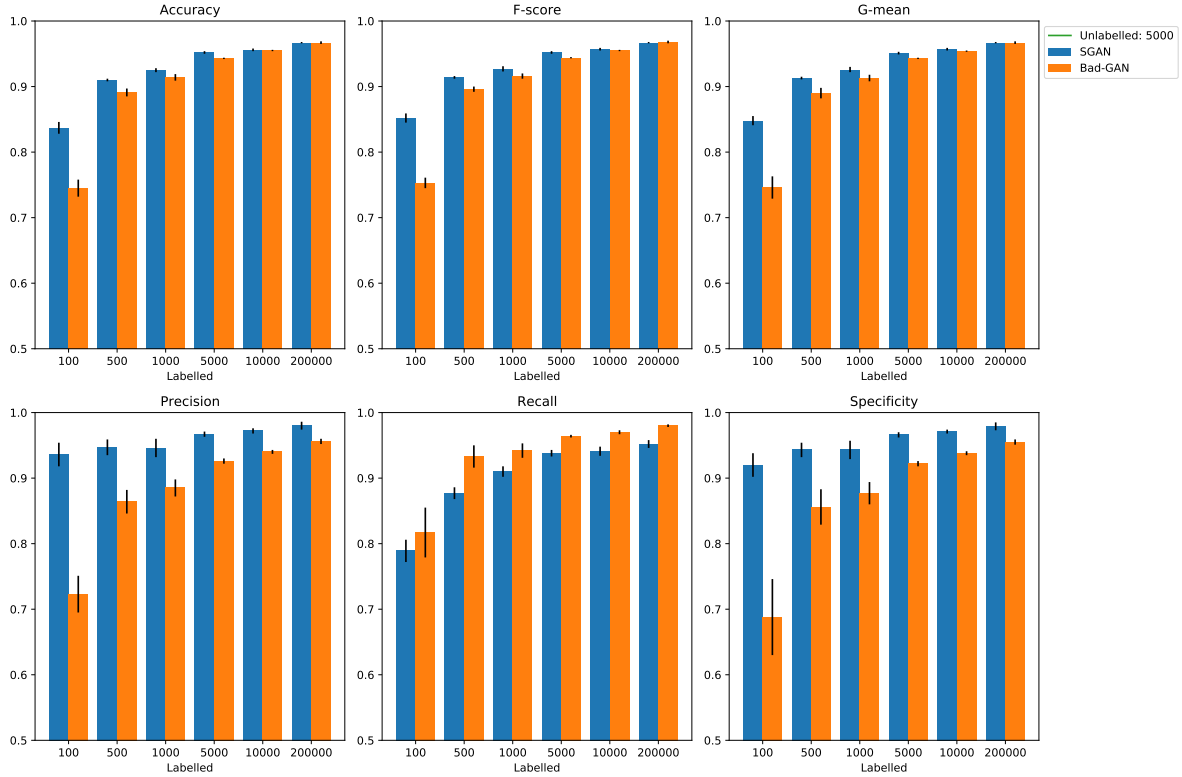

Figure B.6: Performance on the frequency-phase dataset: metrics to test the performance of SGAN and Bad-GAN using 5000 unlabelled frequency-phase examples.

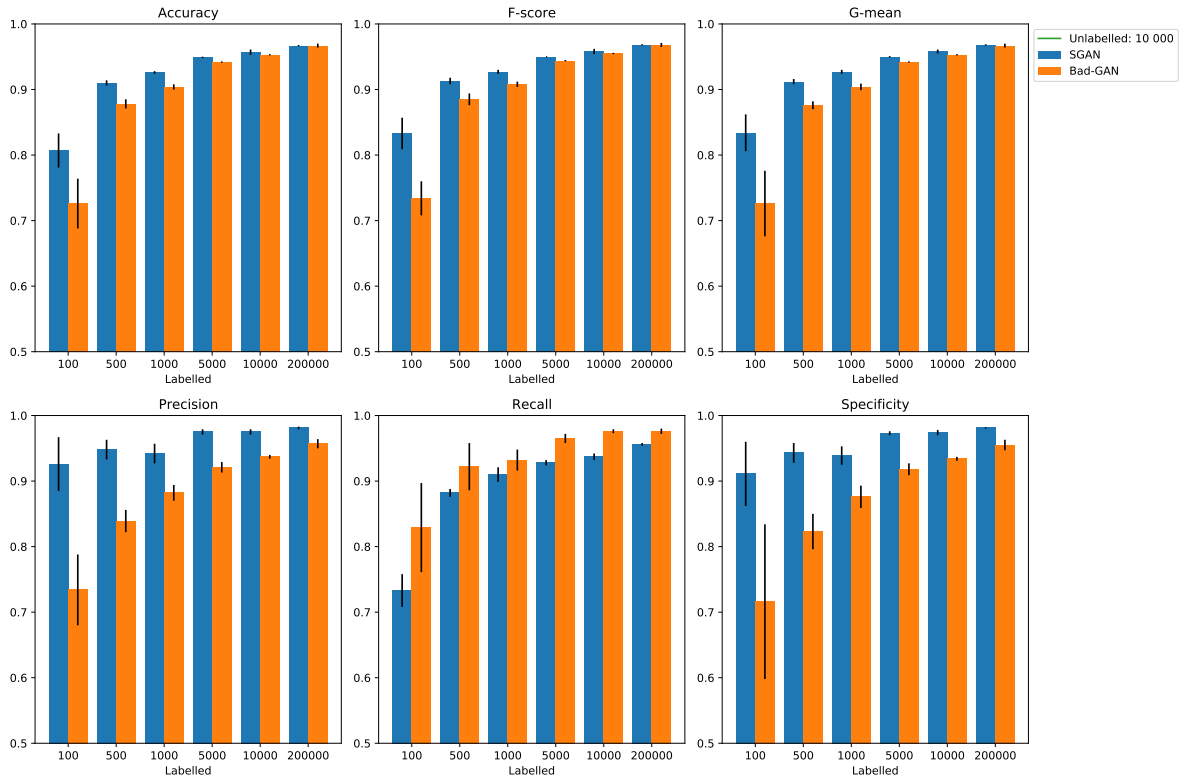

Figure B.7: Performance on the frequency-phase dataset: metrics to test the performance of SGAN and Bad-GAN using 10,000 unlabelled frequency-phase examples.

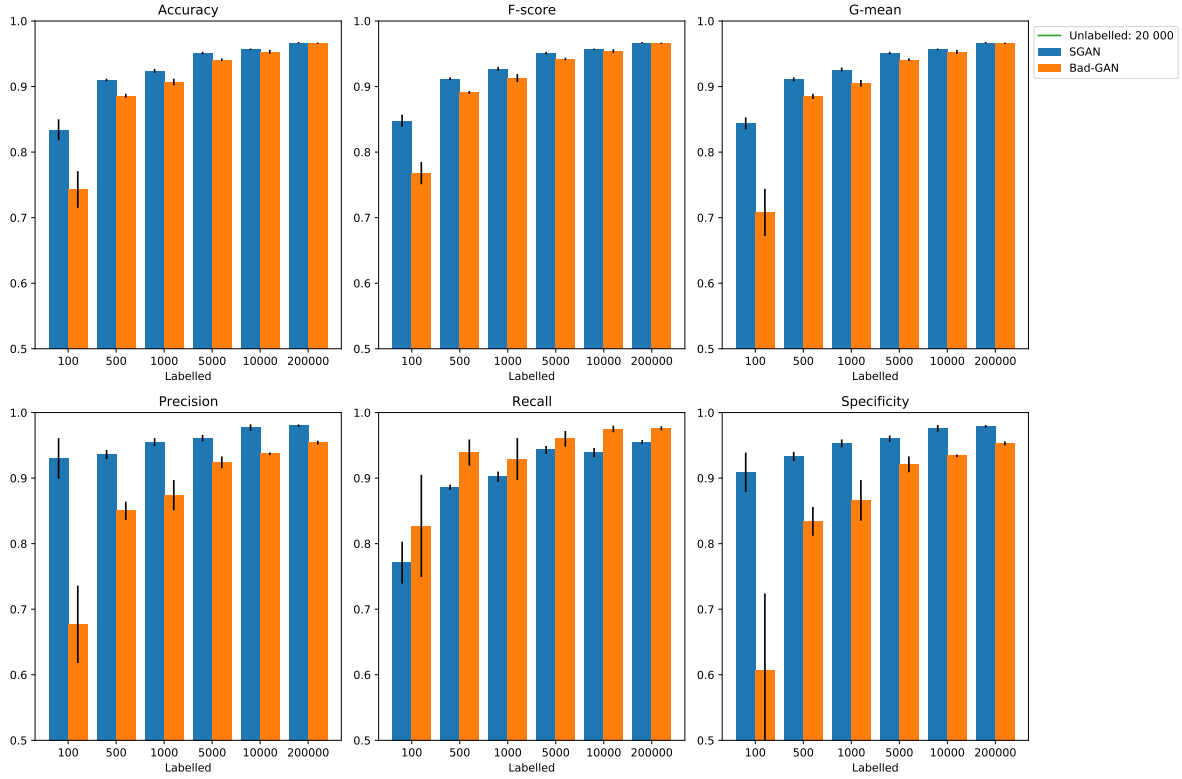

Figure B.8: Performance on the frequency-phase dataset: metrics to test the performance of SGAN and Bad-GAN using 20,000 unlabelled frequency-phase examples.

B.3 Performance of SGAN and Bad-GAN across other metrics score using the pulse-profile dataset

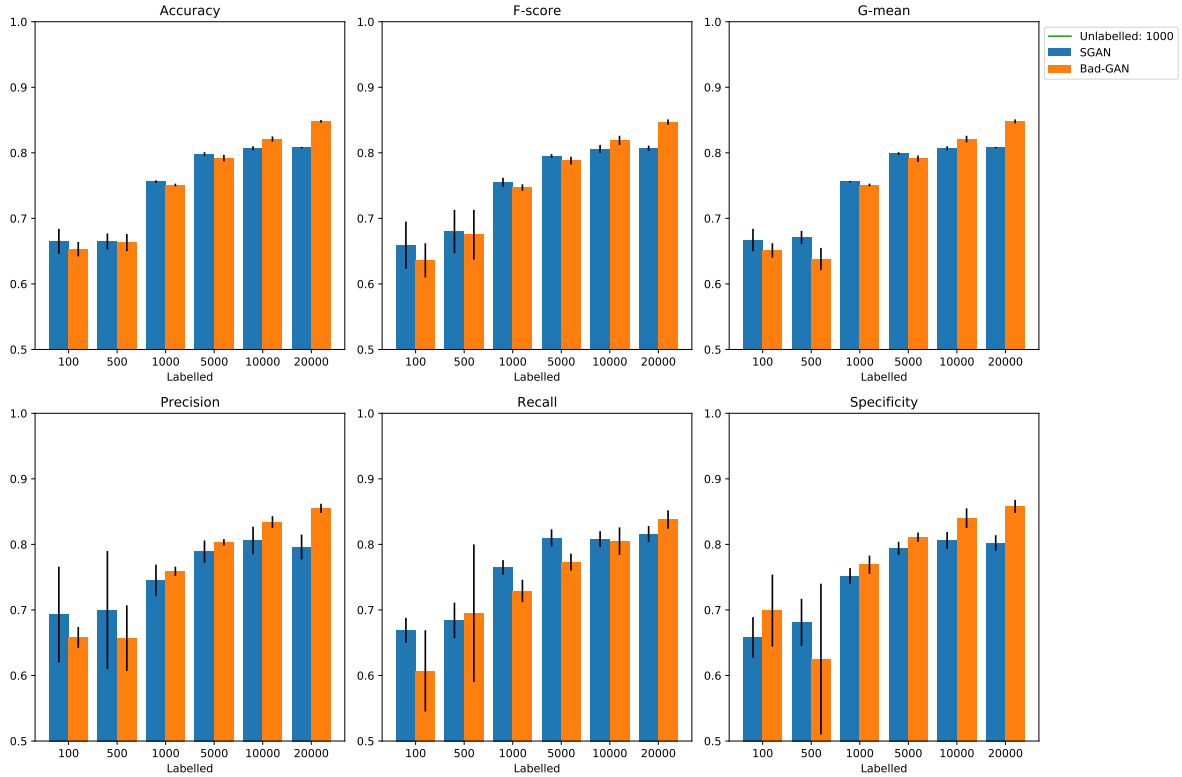

Figure B.9: Performance on the pulse-profile dataset: metrics to test the performance of SGAN and Bad-GAN using 1000 unlabelled pulse-profile examples.

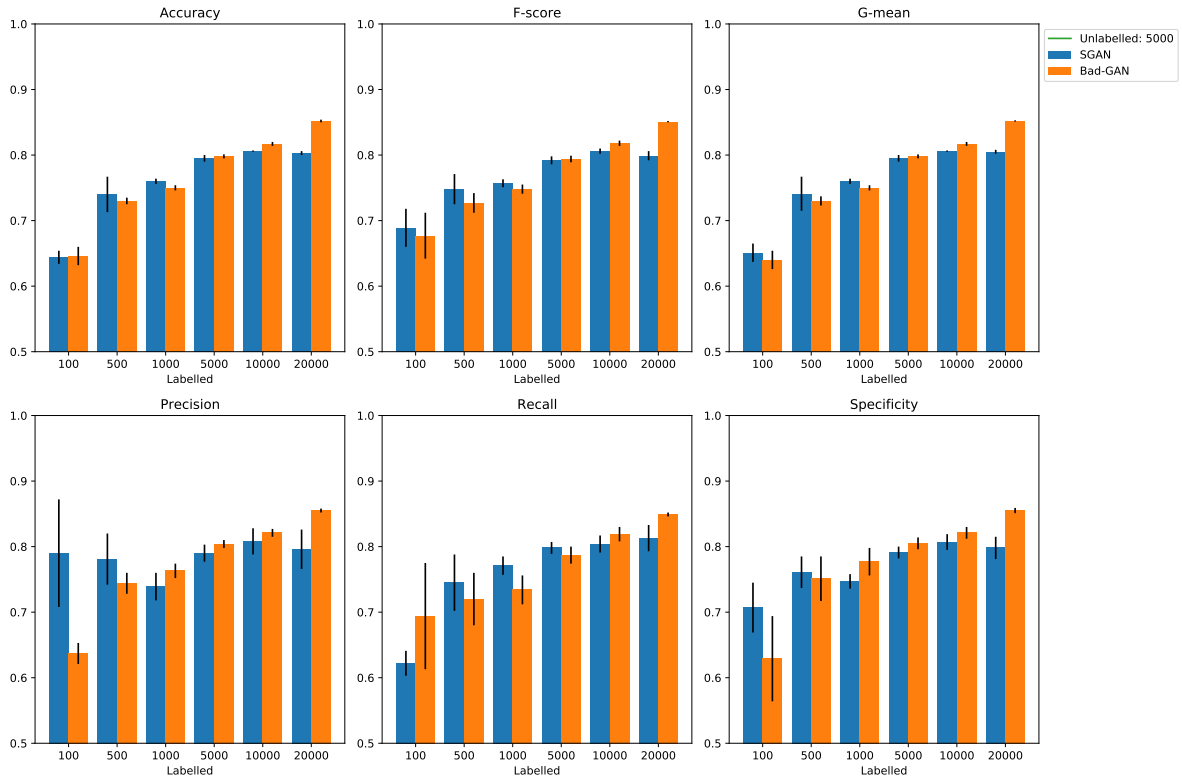

Figure B.10: Performance on the pulse-profile dataset: metrics to test the performance of SGAN and Bad-GAN using 5000 unlabelled pulse-profile examples.

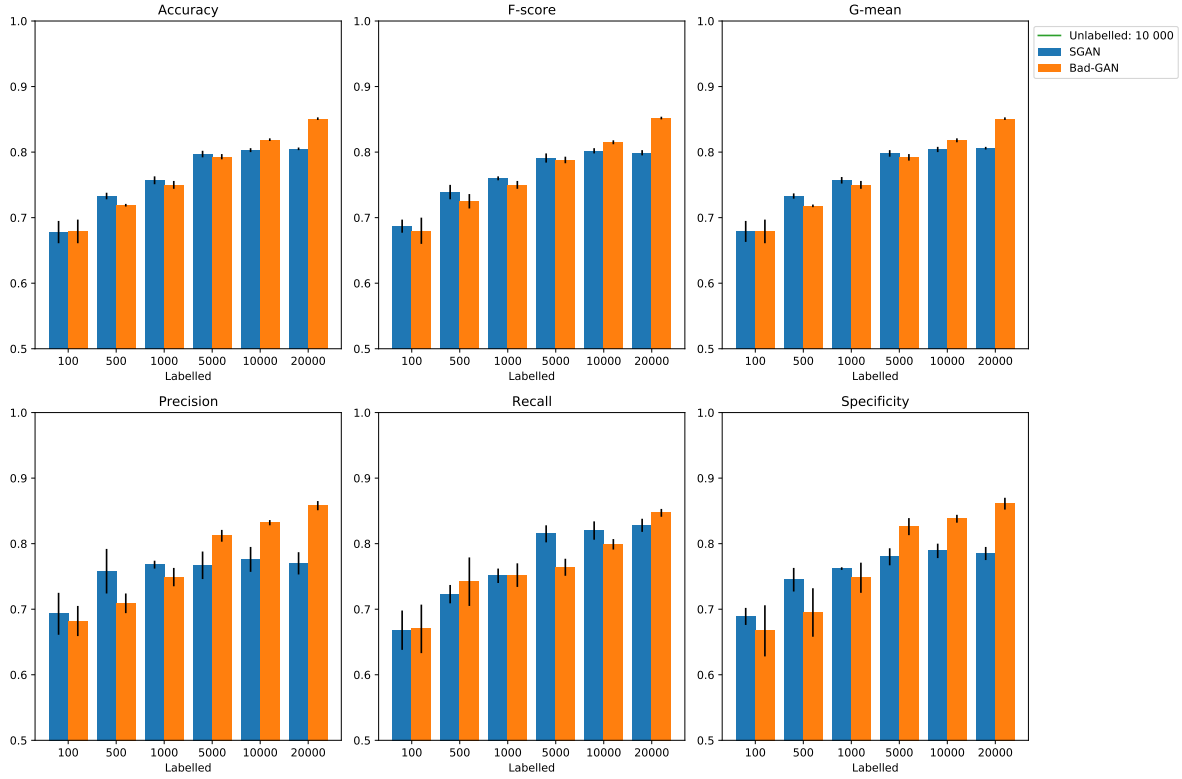

Figure B.11: Performance on the pulse-profile dataset: metrics to test the performance of SGAN and Bad-GAN using 5000 unlabelled pulse-profile examples.

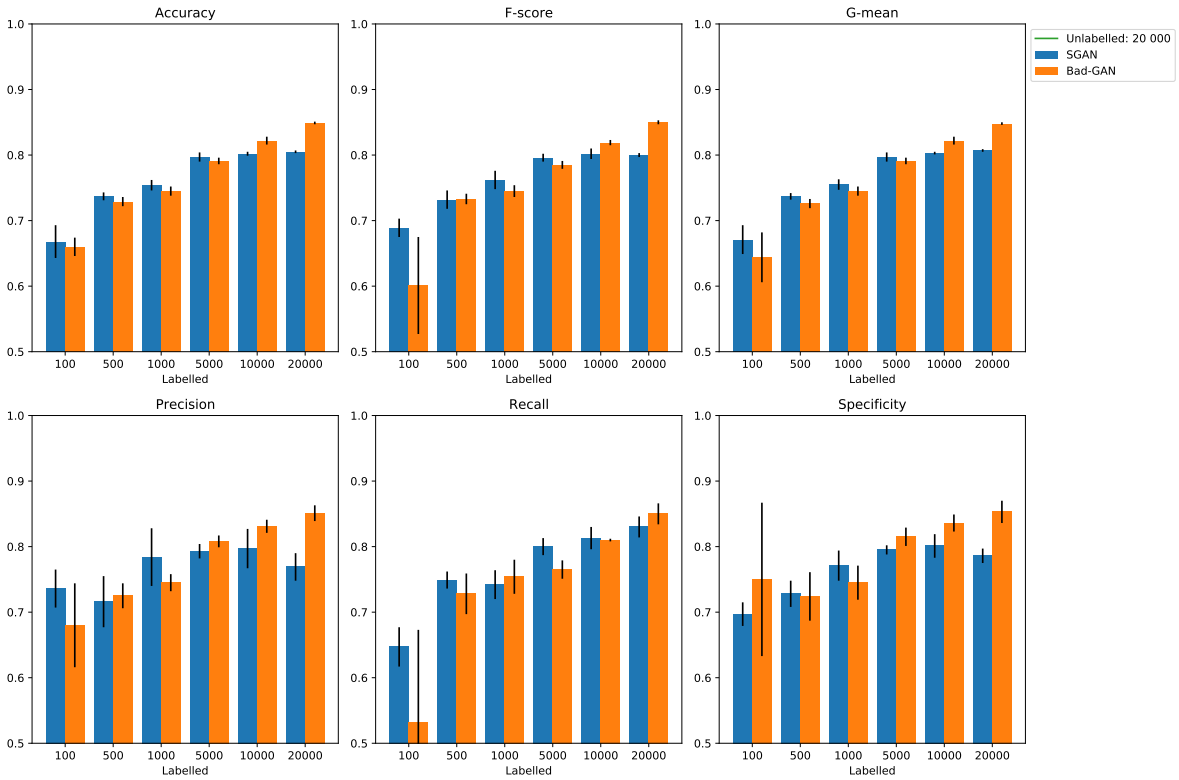

Figure B.12: Performance on the pulse-profile dataset: metrics to test the performance of SGAN and Bad-GAN using 20,000 unlabelled pulse-profile examples.

B.4 Performance of SGAN and Bad-GAN across other metrics score using the time-phase dataset

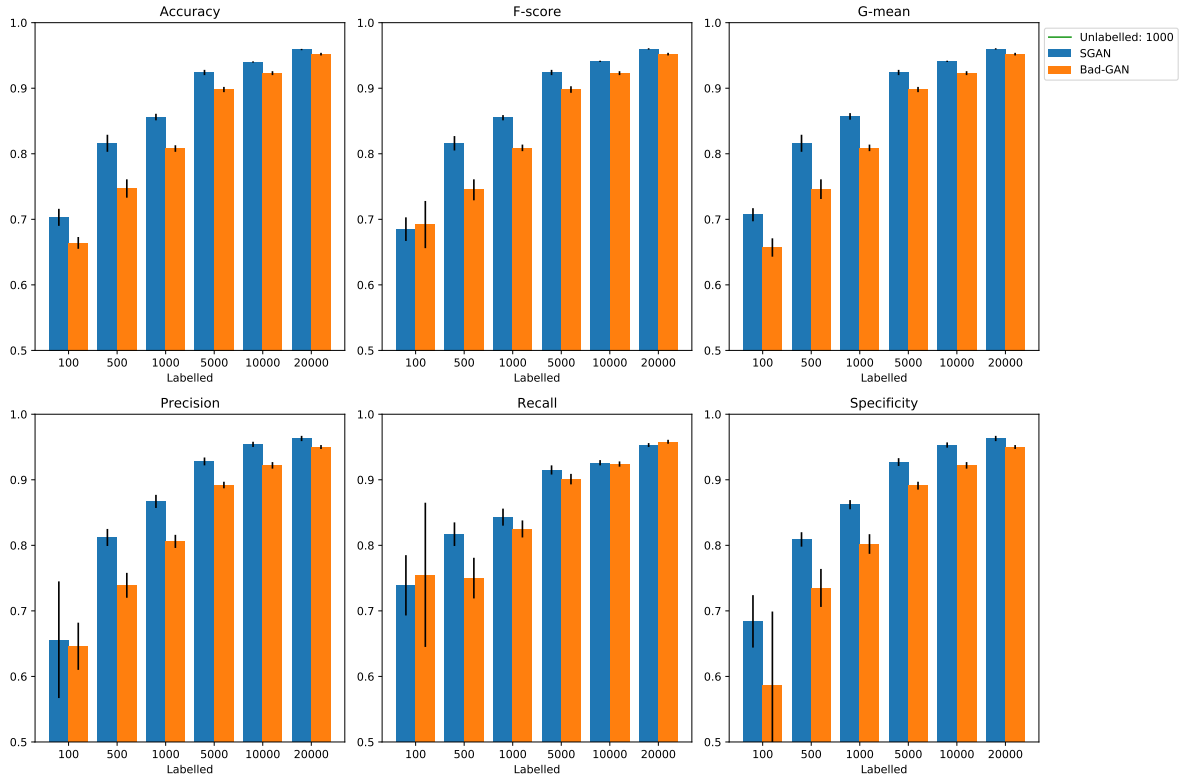

Figure B.13: Performance on the time-phase dataset: metrics to test the performance of SGAN and Bad-GAN using 1000 unlabelled time-phase examples.

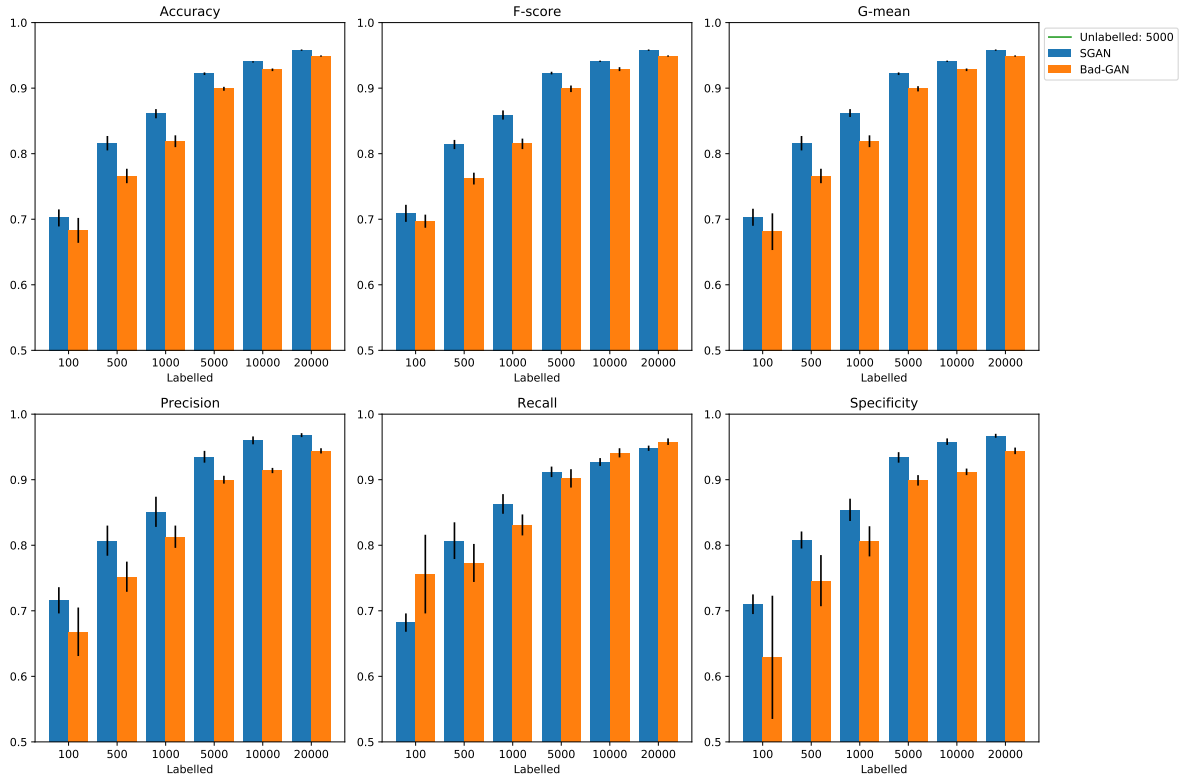

Figure B.14: Performance on the time-phase dataset: metrics to test the performance of SGAN and Bad-GAN using 5000 unlabelled time-phase examples.

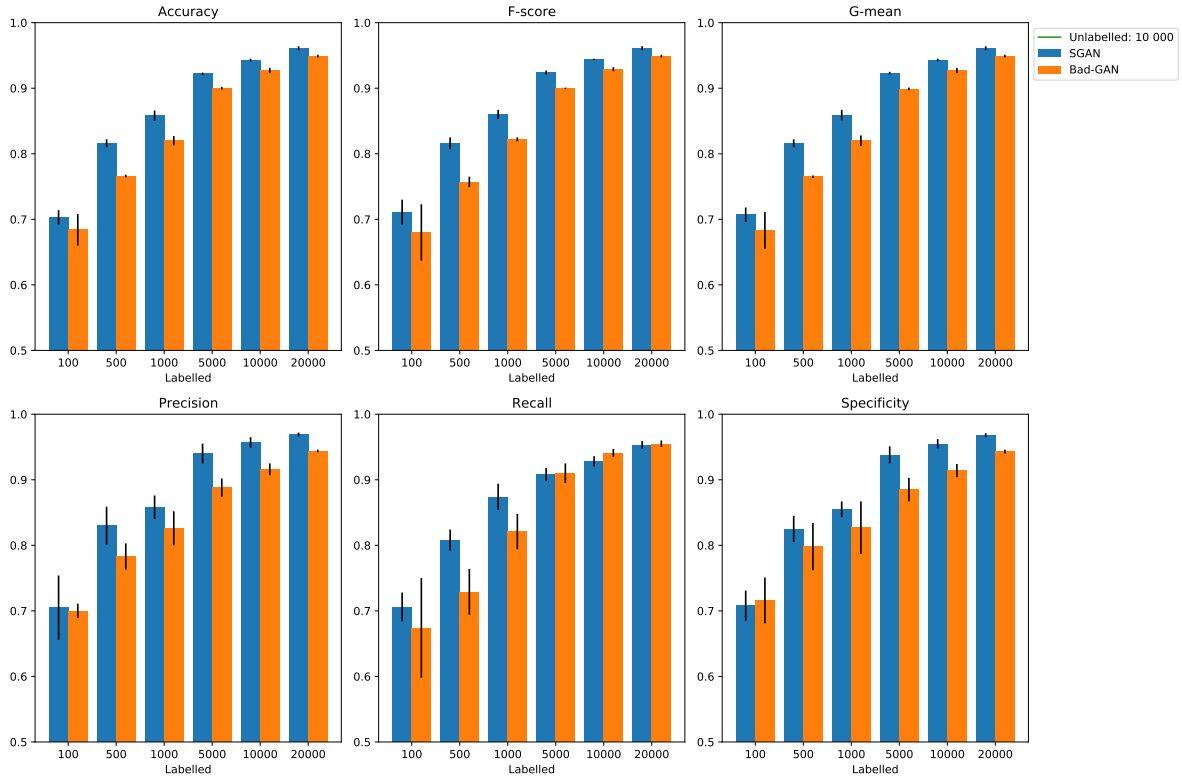

Figure B.15: Performance on the time-phase dataset: metrics to test the performance of SGAN and Bad-GAN using 10,000 unlabelled time-phase examples.

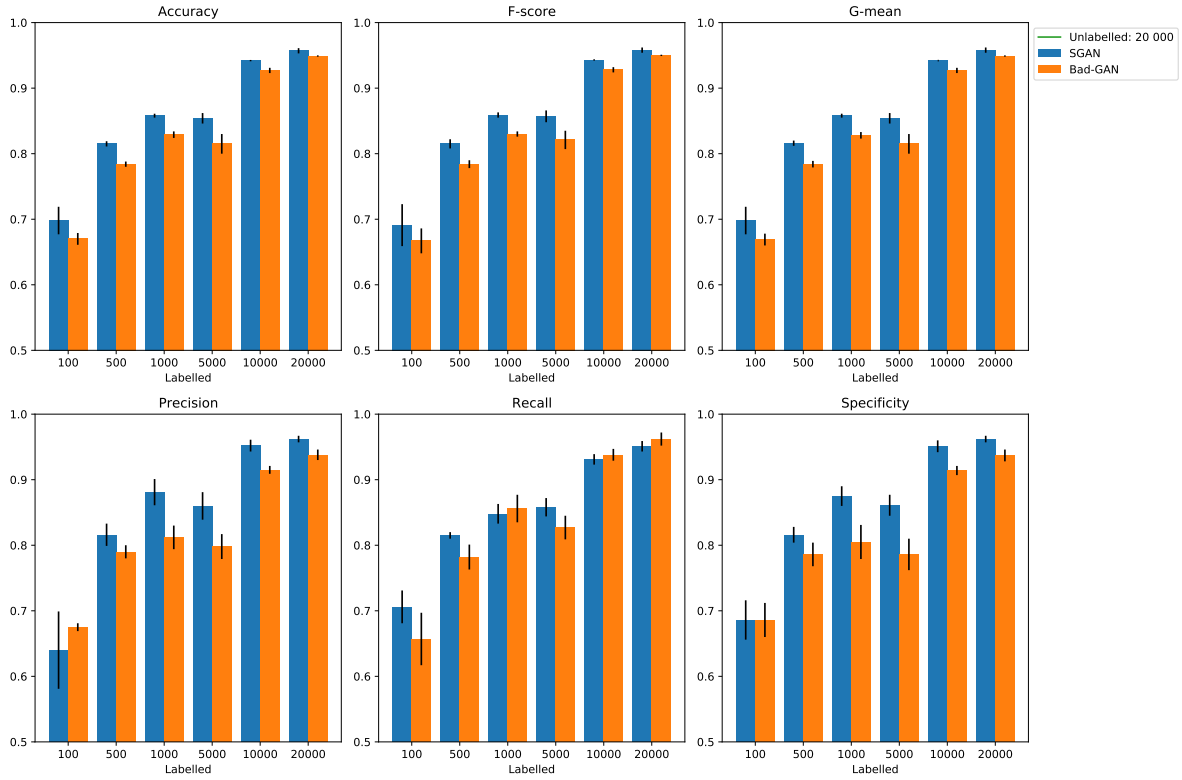

Figure B.16: Performance on the time-phase dataset: metrics to test the performance of SGAN and Bad-GAN using 20,000 unlabelled time-phase examples.

B.5 SGAN performance across various metric scores.

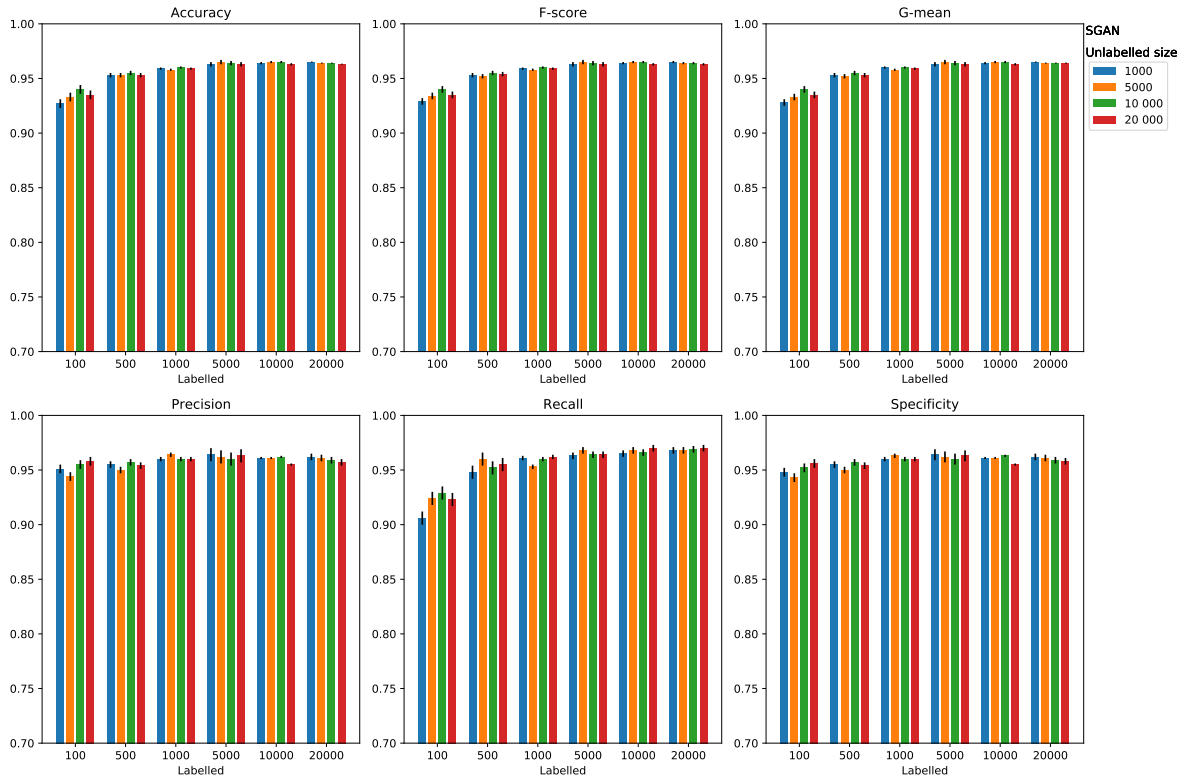

Figure B.17: SGAN performance on the DM-curve dataset as a function of the number of unlabelled examples.

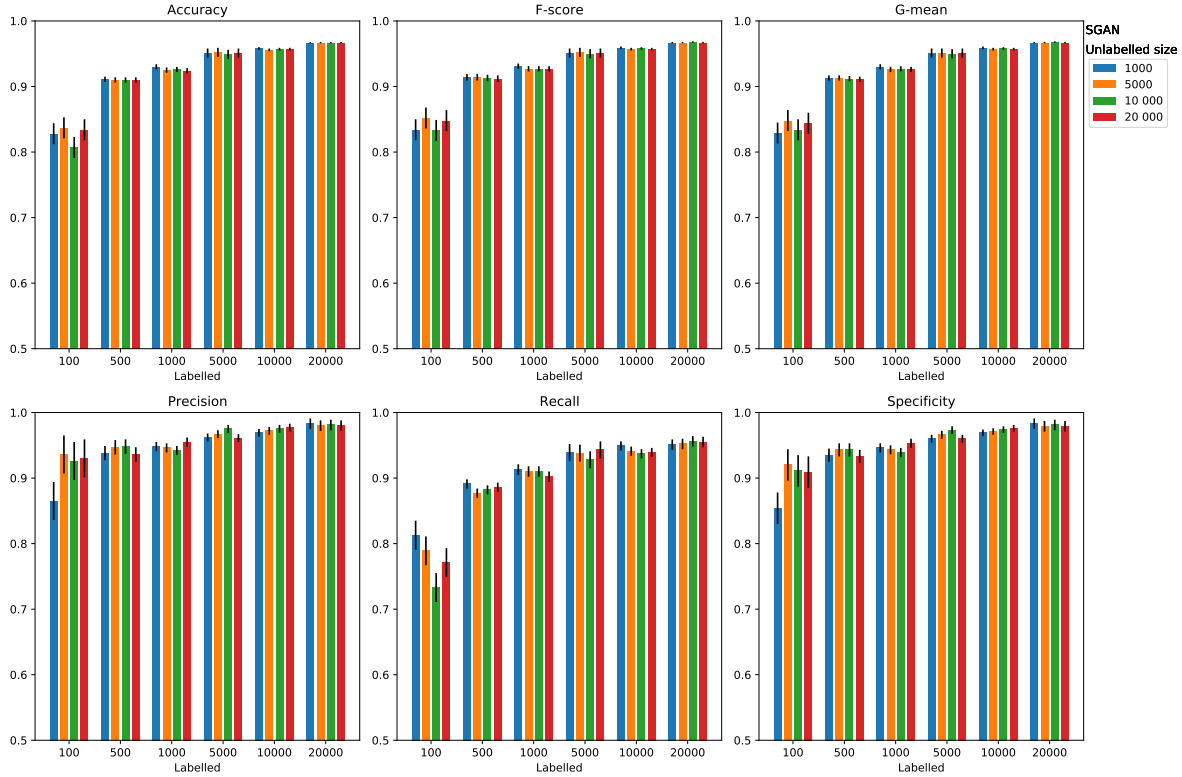

Figure B.18: SGAN performance on the frequency-phase dataset as a function of the number of unlabelled examples.

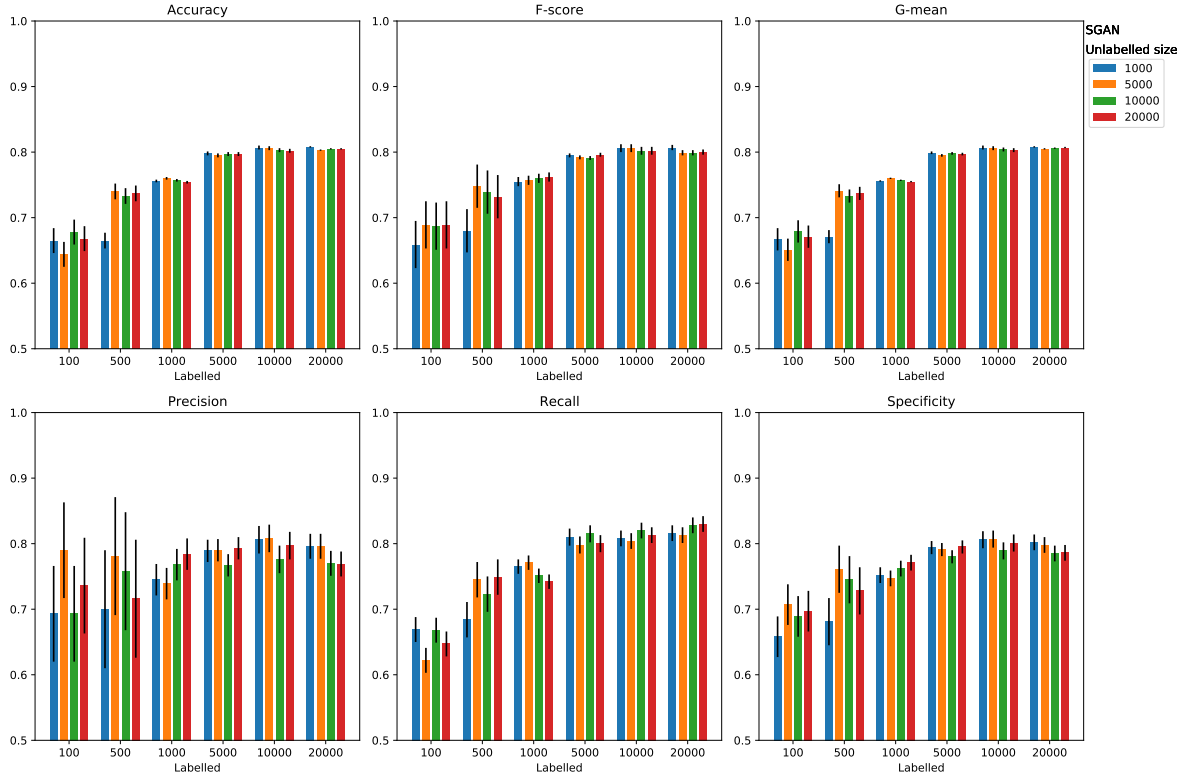

Figure B.19: SGAN performance on the pulse-profile dataset as a function of the number of unlabelled examples.

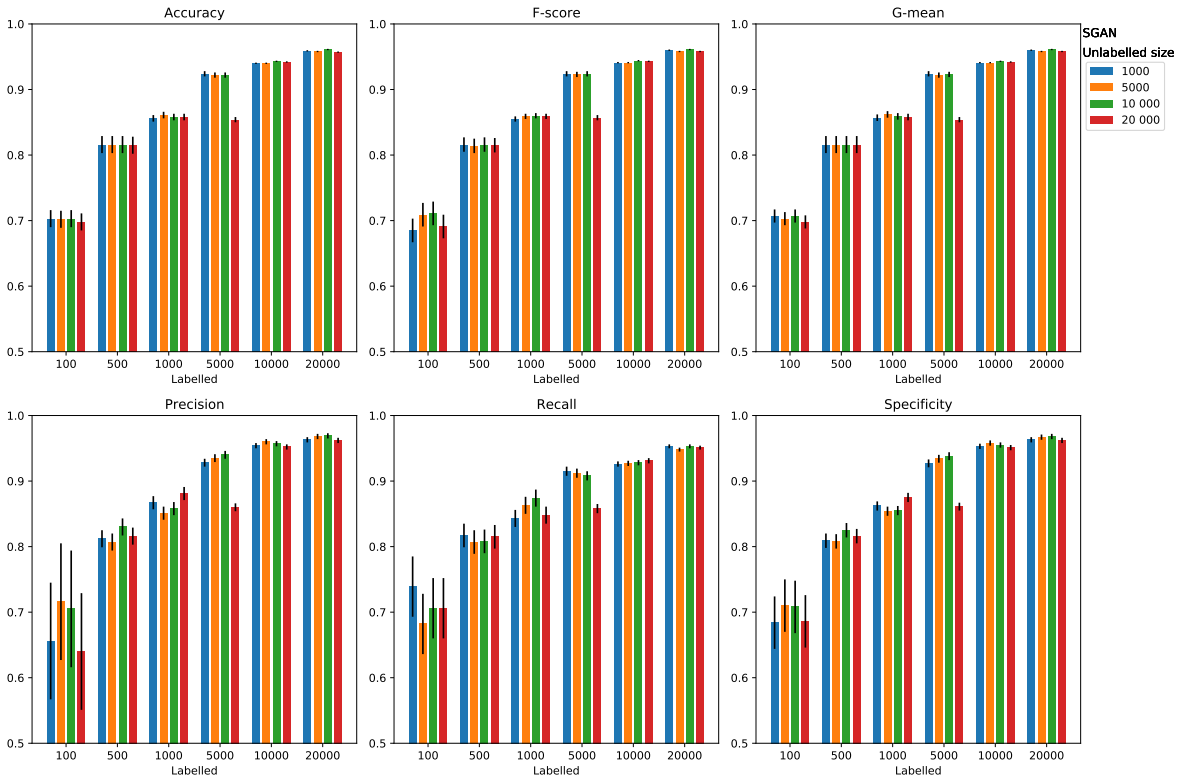

Figure B.20: SGAN performance on the time-phase dataset as a function of the number of unlabelled examples.

B.6 Bad-GAN performance across various metric scores.

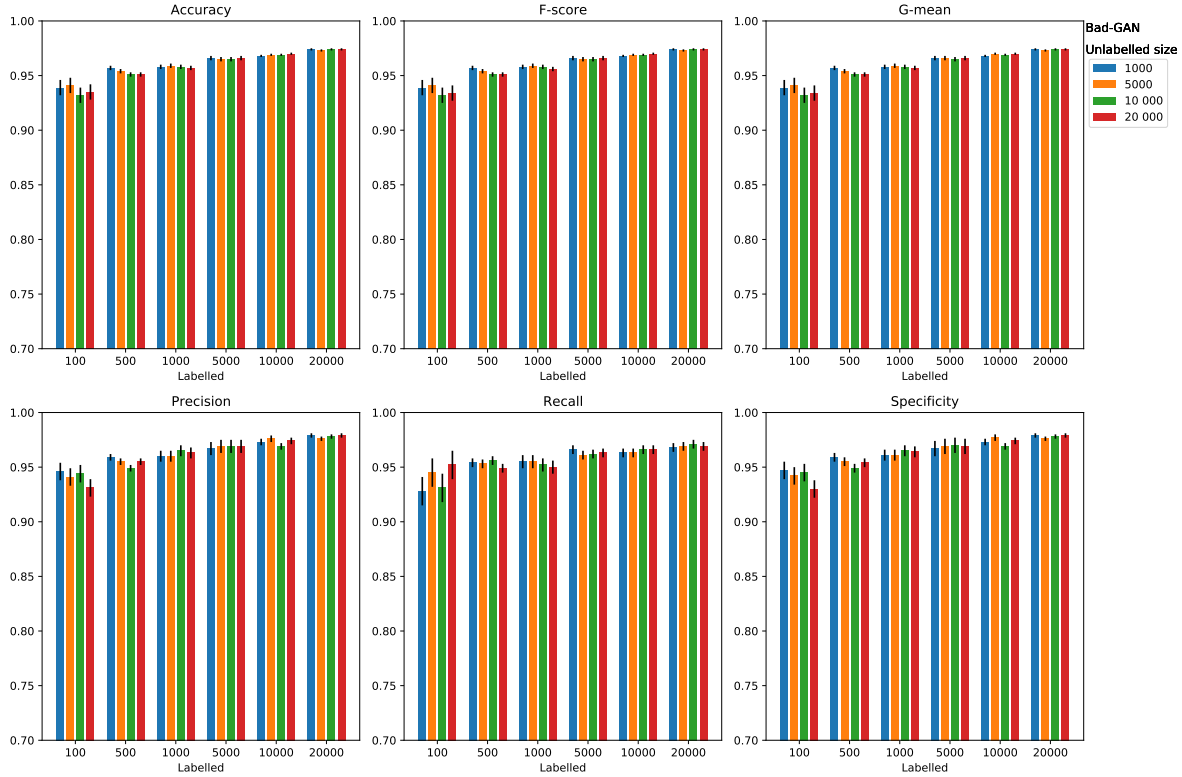

Figure B.21: Bad-GAN performance on the DM-curve dataset as a function of the number of unlabelled examples.

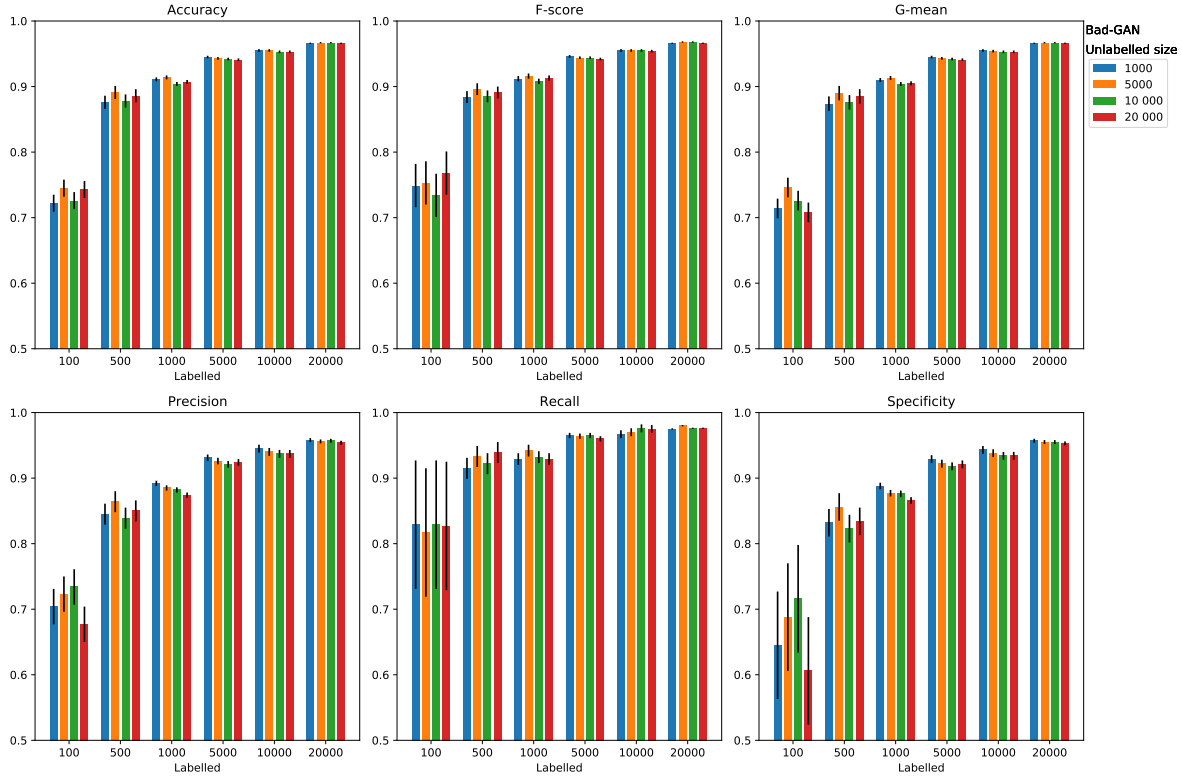

Figure B.22: Bad-GAN performance on the frequency-phase dataset as a function of the number of unlabelled examples.

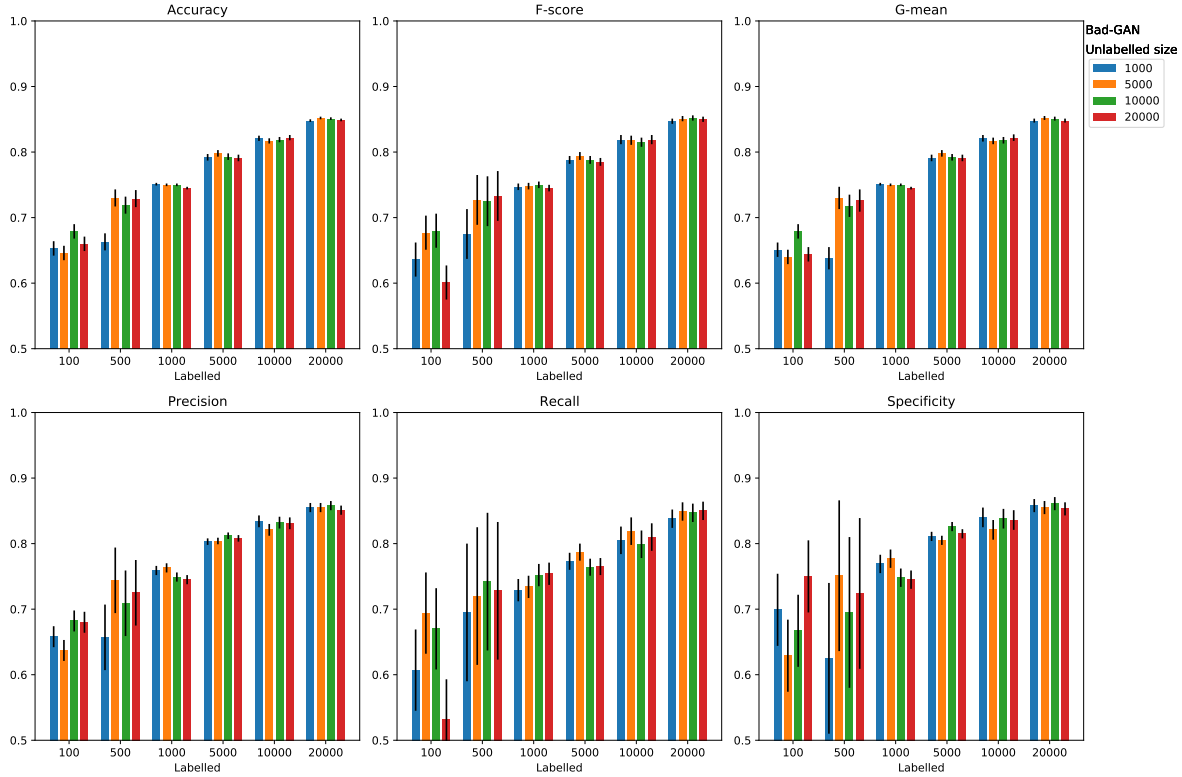

Figure B.23: Bad-GAN performance on the pulse-profile dataset as a function of the number of unlabelled examples.

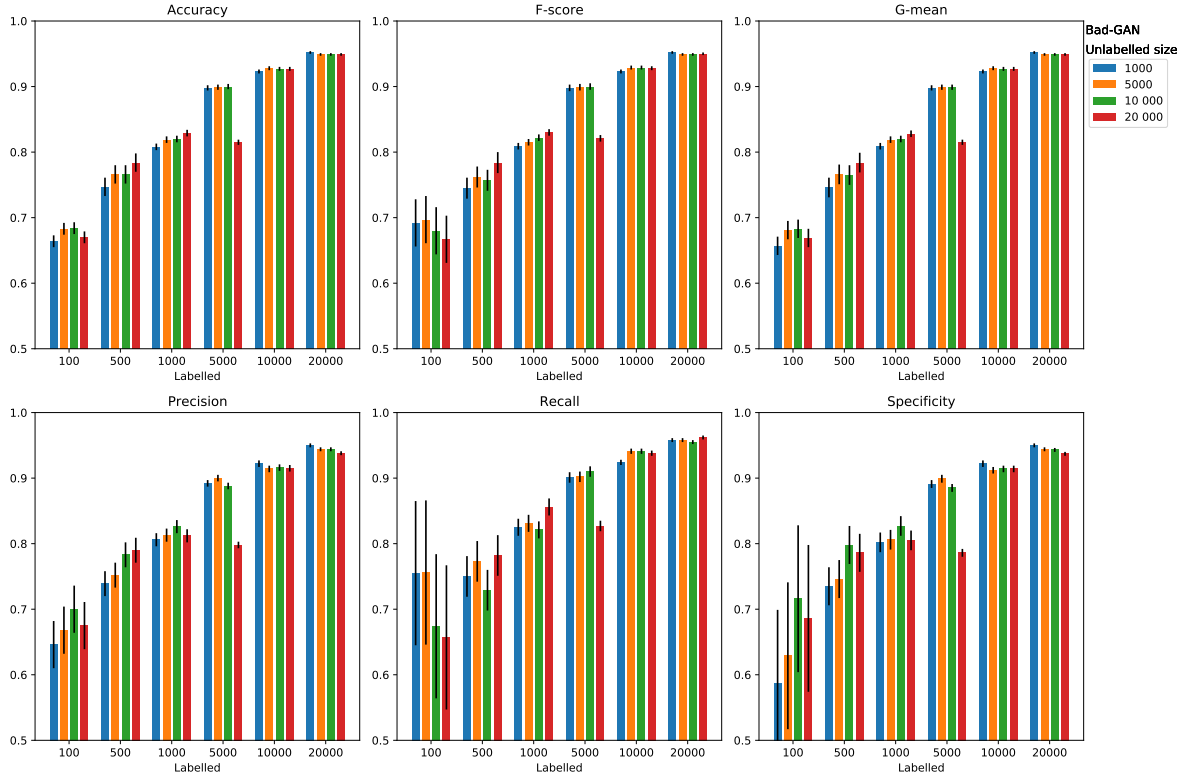

Figure B.24: Bad-GAN performance on the time-phase dataset as a function of the number of unlabelled examples.

Appendix C

Combined performance of SGAN and Bad-GAN

C.1 Combined performance of SGAN

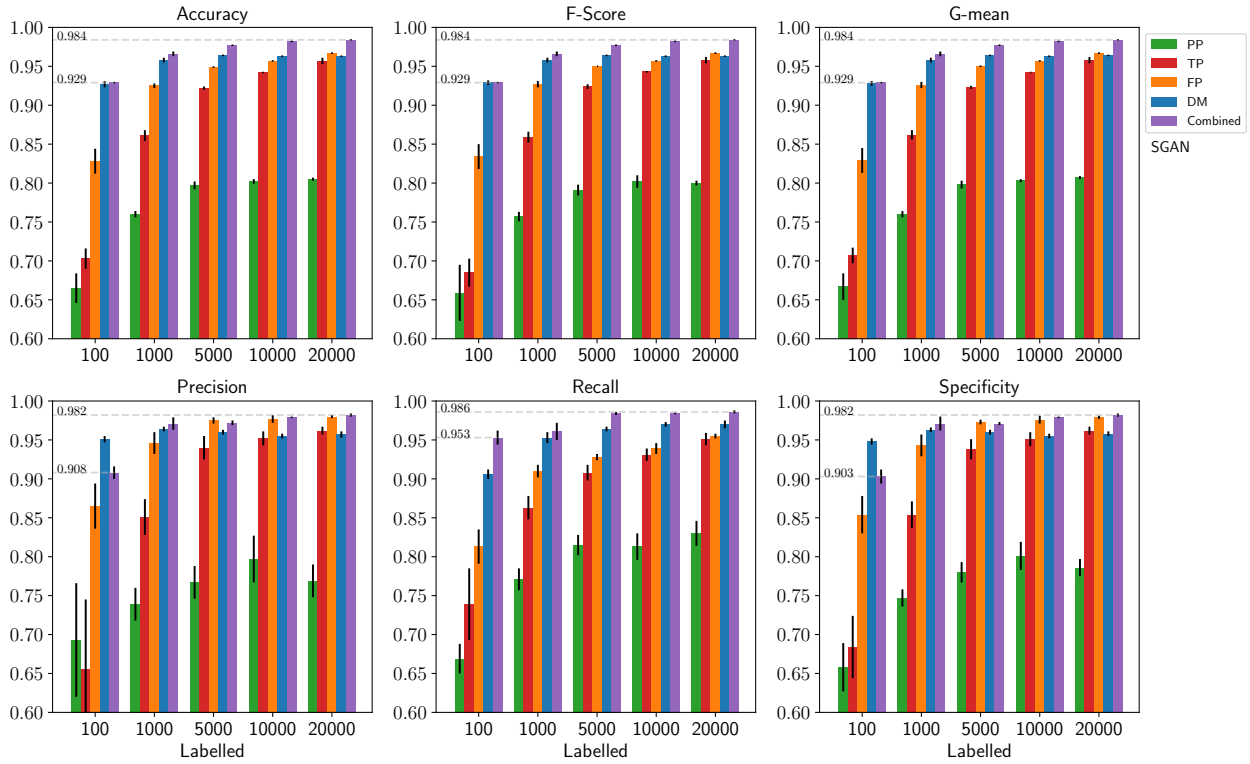

Figure C.1: Combined performance of the SGAN model: the performance of these models trained on pulse-profile (PP, green), time-phase (TP, red), frequency-phase (FP, orange), and DM-curve (DM, blue) was combined to obtain the combined performance (purple). Each bar represents the median value of five performance scores, while the black vertical lines on each bar represent standard deviations (errors) of the scores. Each labelled size, 100, 1000, 5000, 10000, and 20000 corresponds to a model training with 1000, 5000, 10000, 20000, and 20000 unlabelled examples respectively. The grey dashed horizontal lines represent the performance of the combined model on 100 and 20000 labelled examples.

C.2 Combined performance of Bad-GAN

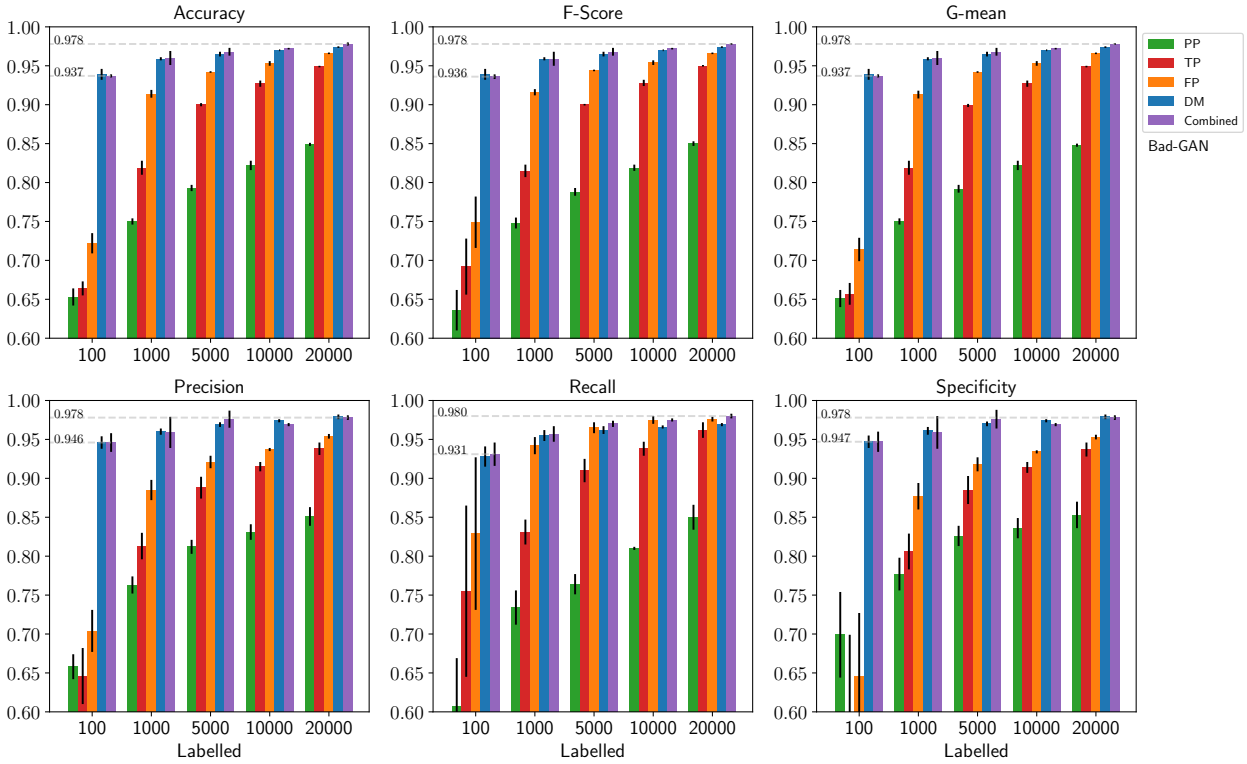

Figure C.2: Combined performance of the Bad-GAN model: the performance of these models trained on pulse-profile (PP, green), time-phase (TP, red), frequency-phase (FP, orange), and DM-curve (DM, blue) was combined to obtain the combined performance (purple). Each bar represents the median value of five performance scores, while the black vertical lines on each bar represent standard deviations (errors) of the scores. Each labelled size, 100, 1000, 5000, 10000, and 20000 corresponds to a model training with 1000, 5000, 10000, 20000, and 20000 unlabelled examples respectively. The grey dashed horizontal lines represent the performance of the combined model on 100 and 20000 labelled examples.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. 2016, in 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), 265–283

Ando, S. & Huang, C.-Y. 2017, arXiv e-prints, arXiv:1704.07515

Annala, E., Gorda, T., Kurkela, A., Nättilä, J., & Vuorinen, A. 2020, Nature Physics, 16, 907

Arjovsky, M., Chintala, S., & Bottou, L. 2017, arXiv e-prints, arXiv:1701.07875

Baade, W. & Zwicky, F. 1934, Physical Review, 46, 76

Balakrishnan, V., Champion, D., Barr, E., Kramer, M., Sengar, R., & Bailes, M. 2021, MNRAS, 505, 1180

Barchi, P., de Carvalho, R., Rosa, R., Sautter, R., Soares-Santos, M., Marques, B., Clua, E., Gonçalves, T., de Sá-Freitas, C., & Moura, T. 2020, Astronomy and Computing, 30, 100334

Baron, D. 2019, arXiv e-prints, arXiv:1904.07248

Barr, E. 2020, Peasoup: C++/CUDA GPU pulsar searching library

Barr, E. D., Champion, D. J., Kramer, M., Eatough, R. P., Freire, P. C. C., Karuppusamy, R., Lee, K. J., Verbiest, J. P. W., Bassa, C. G., Lyne, A. G., Stappers, B., Lorimer, D. R., & Klein, B. 2013, MNRAS, 435, 2234

Bates, S. D., Bailes, M., Barsdell, B. R., Bhat, N. D. R., Burgay, M., Burke-Spolaor, S., Champion, D. J., Coster, P., D'Amico, N., Jameson, A., et al. 2012, MNRAS, 427, 1052

Batista, G. E. A. P. A., Prati, R. C., & Monard, M. C. 2004, SIGKDD Explor. Newsl., 6, 20–29

Bäuerle, A., van Onzenoodt, C., & Ropinski, T. 2019, arXiv e-prints, arXiv:1902.04394

Baym, G. & Pethick, C. 1979, ARA&A, 17, 415

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. 2011, in Advances in Neural Information Processing Systems, ed. J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, & K. Q. Weinberger, Vol. 24 (Curran Associates, Inc.)

Bergstra, J. & Bengio, Y. 2012, Journal of Machine Learning Research, 13, 281

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., & Cox, D. D. 2015, Computational Science & Discovery, 8, 014008

Bergstra, J., Yamins, D., & Cox, D. D. 2013, in Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML'13 (JMLR.org), I–115–I–123

Bethapudi, S. & Desai, S. 2018, Astronomy and Computing, 23, 15

Bhat, A. & Malyshev, D. 2021, arXiv e-prints, arXiv:2102.07642

Bhattacharya, D. & van den Heuvel, E. P. J. 1991, Phys. Rep., 203, 1

Bishop, C. M. 1995, Neural Networks for Pattern Recognition (USA: Oxford University Press, Inc.)

—. 2006, Pattern Recognition and Machine Learning (Information Science and Statistics) (Berlin, Heidelberg: Springer-Verlag)

Bonaccorso, G. 2017, Machine Learning Algorithms: A Reference Guide to Popular Algorithms for Data Science and Machine Learning (Packt Publishing)

Boslaugh, S. 2012, Statistics in a Nutshell, 2nd Edition (O'Reilly Media, Incorporated)

Braga-Neto, U. M. & Dougherty, E. R. 2020, IEEE Signal Processing Magazine, 37, 118

Breiman, L. 1996a, Mach. Learn., 24, 123

—. 1996b, Bias, variance, and arcing classifiers, Tech. Rep. 460, Statistics Department, University of California at Berkeley

—. 1996c, Mach. Learn., 24, 49

—. 2001, Mach. Learn., 45, 5

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. 1984, Classification and Regression Trees (Wadsworth)

Brock, A., Donahue, J., & Simonyan, K. 2018, arXiv e-prints, arXiv:1809.11096

Buda, M., Maki, A., & Mazurowski, M. A. 2017, arXiv e-prints, arXiv:1710.05381

Byrd, J. & Lipton, Z. 2019, in Proceedings of Machine Learning Research, Vol. 97, Proceedings of the 36th International Conference on Machine Learning, ed. K. Chaudhuri & R. Salakhutdinov (PMLR), 872–881

Casini, H. & Montemayor, R. 1998, ApJ, 503, 374

Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L., & Lopez, A. 2020, Neurocomputing, 408, 189

Chapelle, O., Schlkopf, B., & Zien, A. 2010, Semi-Supervised Learning, 1st edn. (The MIT Press)

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. 2011, arXiv e-prints, arXiv:1106.1813

Chen, T. & Guestrin, C. 2016, arXiv e-prints, arXiv:1603.02754

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., & Abbeel, P. 2016, arXiv e-prints, arXiv:1606.03657

Chollet, F. 2017, Deep Learning with Python (Manning)

Chollet, F. et al. 2015, Keras

Clarke, A. O., Scaife, A. M. M., Greenhalgh, R., & Griguta, V. 2020, A&A, 639, A84

Clifton, T. R. & Lyne, A. G. 1986, Nature, 320, 43

Cordes, J. M., Freire, P. C. C., Lorimer, D. R., Camilo, F., Champion, D. J., Nice, D. J., Ramachand ran, R., Hessels, J. W. T., Vlemmings, W., van Leeuwen, J., et al. 2006, ApJ, 637, 446

Cordes, J. M. & Shannon, R. M. 2010, arXiv e-prints, arXiv:1010.3785

Cristianini, N. & Shawe-Taylor, J. 2000, An Introduction to Support Vector Machines and Other Kernel-based Learning Methods (Cambridge University Press)

Dai, Z., Yang, Z., Yang, F., Cohen, W. W., & Salakhutdinov, R. 2017, arXiv e-prints, arXiv:1705.09783

Denton, E., Chintala, S., Szlam, A., & Fergus, R. 2015, arXiv e-prints, arXiv:1506.05751

Devlin, J., Chang, M., Lee, K., & Toutanova, K. 2019, in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), ed. J. Burstein, C. Doran, & T. Solorio (Association for Computational Linguistics), 4171–4186

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. 2018, arXiv e-prints, arXiv:1810.04805

Dorn-Wallenstein, T. Z., Davenport, J. R. A., Huppenkothen, D., & Levesque, E. M. 2021, ApJ, 913, 32

Eatough, R. P., Molkenthin, N., Kramer, M., Noutsos, A., Keith, M. J., Stappers, B. W., & Lyne, A. G. 2010, MNRAS, 407, 2443

Efron, B. & Tibshirani, R. 1986, Statistical Science, 1, 54

Faucher-Giguère, C.-A. & Kaspi, V. M. 2006, ApJ, 643, 332

Faulkner, A. J., Stairs, I. H., Kramer, M., Lyne, A. G., Hobbs, G., Possenti, A., Lorimer, D. R., Manchester, R. N., McLaughlin, M. A., D'Amico, N., Camilo, F., & Burgay, M. 2004, MNRAS, 355, 147

Forsyth, D. A. 2019, Applied Machine Learning (Springer)

Freund, Y. & Schapire, R. E. 1996, in Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996, ed. L. Saitta (Morgan Kaufmann), 148–156

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

Friedman, J. H. 2001, The Annals of Statistics, 29, 1189

Gayatri, K., Divya Kanti, R., Sekhar Rao Rayavarapu, V. C., Sridhar, B., D., & Rama Gowri Bobbili, V. 2021, in Journal of Physics Conference Series, Vol. 1804, Journal of Physics Conference Series, 012160

Geman, S., Bienenstock, E., & Doursat, R. 1992, Neural Comput., 4, 1

Gholami, R. & Fakhari, N. 2017, in Handbook of Neural Computation, ed. P. Samui, S. Sekhar, & V. E. Balas (Academic Press), 515–535

Ghosh, P. 2007, Rotation and Accretion Powered Pulsars (WORLD SCIENTIFIC)

Golob, A., Sawicki, M., Goulding, A. D., & Coupon, J. 2021, MNRAS, 503, 4136

Goodfellow, I. 2016, arXiv e-prints, arXiv:1701.00160

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. 2014, arXiv e-prints, arXiv:1406.2661

Graves, A. & Jaitly, N. 2014, in Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, ICML'14 (JMLR.org), II–1764–II–1772

Guo, P., Duan, F., Wang, P., Yao, Y., Yin, Q., & Xin, X. 2017, arXiv e-prints, arXiv:1711.10339

Guo, P., Duan, F., Wang, P., Yao, Y., Yin, Q., Xin, X., Li, D., Qian, L., Wang, S., Pan, Z., & Zhang, L. 2019, MNRAS, 490, 5424

Hamil, O., Stone, J. R., Urbanec, M., & Urbancová, G. 2015, Phys. Rev. D, 91, 063007

Han, J., Zhang, D., Cheng, G., Guo, L., & Ren, J. 2015, IEEE Transactions on Geoscience and Remote Sensing, 53, 3325

Harry, I. W., Allen, B., & Sathyaprakash, B. S. 2009, Phys. Rev. D, 80, 104014

Hastie, T., Tibshirani, R., & Friedman, J. 2009, The elements of statistical learning: data mining, inference and prediction, 2nd edn. (Springer)

Hawley, M. S., Cunningham, S. P., Green, P. D., Enderby, P., Palmer, R., Sehgal, S., & O'Neill, P. 2013, IEEE Transactions on Neural Systems and Rehabilitation Engineering, 21, 23

He, H. & Ma, Y. 2013, Imbalanced Learning: Foundations, Algorithms, and Applications, 1st edn. (Wiley-IEEE Press)

He, K., Fan, H., Wu, Y., Xie, S., & Girshick, R. 2020, in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 9726–9735

Hessels, J. W. T., Ransom, S. M., Stairs, I. H., Freire, P. C. C., Kaspi, V. M., & Camilo, F. 2006, in American Astronomical Society Meeting Abstracts, Vol. 207, American Astronomical Society Meeting Abstracts #207, 209.07

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. 2012, arXiv e-prints, arXiv:1207.0580

Hobbs, G. & Dai, S. 2017, National Science Review, 4, 707

Hobbs, G. B., Edwards, R. T., & Manchester, R. N. 2006, MNRAS, 369, 655

Ioffe, S. & Szegedy, C. 2015, arXiv e-prints, arXiv:1502.03167

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. 2016, arXiv e-prints, arXiv:1611.07004

Jiwoong Im, D., Dongjoo Kim, C., Jiang, H., & Memisevic, R. 2016, arXiv e-prints, arXiv:1602.05110

Joachims, T. 1999, in (Morgan Kaufmann), 200–209

Johnston, S. & Karastergiou, A. 2017, MNRAS, 467, 3493

Joshi, N., Kumar, A., Chakraborty, P., & Kala, R. 2015, in 2015 Third International Conference on Image Information Processing (ICIIP), 526–530

Karras, T., Aila, T., Laine, S., & Lehtinen, J. 2017, arXiv e-prints, arXiv:1710.10196

Kaspi, V. M. & Kramer, M. 2016, arXiv e-prints, arXiv:1602.07738

Keane, E. F. & Kramer, M. 2008, MNRAS, 391, 2009

Keith, M. J., Jameson, A., van Straten, W., Bailes, M., Johnston, S., Kramer, M., Possenti, A., Bates, S. D., Bhat, N. D. R., Burgay, M., Burke-Spolaor, S., D'Amico, N., Levin, L., McMahon, P. L., Milia, S., & Stappers, B. W. 2010, MNRAS, 409, 619

Kim, E. J., Brunner, R. J., & Carrasco Kind, M. 2015, MNRAS, 453, 507

Kingma, D. P. & Ba, J. 2014, arXiv e-prints, arXiv:1412.6980

Kingma, D. P. & Welling, M. 2013, arXiv e-prints, arXiv:1312.6114

Kodali, N., Abernethy, J., Hays, J., & Kira, Z. 2017, arXiv e-prints, arXiv:1705.07215

Kramer, M. 2004, Millisecond Pulsars as Tools of Fundamental Physics, ed. S. G. Karshenboim & E. Peik (Berlin, Heidelberg: Springer Berlin Heidelberg), 33–54

Krizhevsky, A. 2009, 32

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., & Bengio, Y. 2007, in Proceedings of the 24th International Conference on Machine Learning, ICML '07 (New York, NY, USA: Association for Computing Machinery), 473–480

Lecouat, B., Foo, C.-S., Zenati, H., & Chandrasekhar, V. R. 2018, arXiv e-prints, arXiv:1805.08957

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. 1998, Proceedings of the IEEE, 86, 2278

Lee, K. J., Stovall, K., Jenet, F. A., Martinez, J., Dartez, L. P., Mata, A., Lunsford, G., Cohen, S., Biwer, C. M., Rohr, M., et al. 2013, MNRAS, 433, 688

Levin, L., Armour, W., Baffa, C., Barr, E., Cooper, S., Eatough, R., Ensor, A., Giani, E., Karastergiou, A., Karuppusamy, R., & et al. 2017, Proceedings of the International Astronomical Union, 13, 171–174

Li, W., Wang, Z., Li, J., Polson, J., Speier, W., & Arnold, C. 2019, ArXiv, abs/1905.06484

Lin, H., Li, X., & Luo, Z. 2020a, MNRAS, 493, 1842

Lin, H., Li, X., & Zeng, Q. 2020b, ApJ, 899, 104

Lorimer, D. R. 2001, Living Reviews in Relativity, 4, 5

—. 2005, Living Reviews in Relativity, 8, 7

—. 2008, Living Reviews in Relativity, 11, 8

Lorimer, D. R., Bailes, M., McLaughlin, M. A., Narkevic, D. J., & Crawford, F. 2007, Science, 318, 777

Lyne, A. & Graham-Smith, F. 2012, 2 Neutron stars, 4th edn., Cambridge Astrophysics (Cambridge University Press), 16–26

Lyon, R. J., Stappers, B. W., Cooper, S., Brooke, J. M., & Knowles, J. D. 2016, MNRAS, 459, 1104

López-Antequera, M., Leyva Vallina, M., Strisciuglio, N., & Petkov, N. 2019, IEEE Access, 7, 66157

Maclin, R. & Opitz, D. 2011, arXiv e-prints, arXiv:1106.0257

Manchester, R. N., Lyne, A. G., Camilo, F., Bell, J. F., Kaspi, V. M., D'Amico, N., McKay, N. P. F., Crawford, F., Stairs, I. H., Possenti, A., Kramer, M., & Sheppard, D. C. 2001, MNRAS, 328, 17

Manchester, R. N., Lyne, A. G., Taylor, J. H., Durdin, J. M., Large, M. I., & Little, A. G. 1978, MNRAS, 185, 409

Mao, X., Li, Q., Xie, H., Lau, R. Y. K., Wang, Z., & Smolley, S. P. 2016, arXiv e-prints, arXiv:1611.04076

Melotti, G., Premebida, C., Bird, J. J., Faria, D. R., & Gonçalves, N. 2020, arXiv e-prints, arXiv:2005.14565

Mitchell, T. M. 1997, Machine Learning, 1st edn. (USA: McGraw-Hill, Inc.)

Morello, V., Barr, E. D., Bailes, M., Flynn, C. M., Keane, E. F., & van Straten, W. 2014, MNRAS, 443, 1651

Nalbantov, G. & Ivanov, S. 2019, arXiv e-prints, arXiv:1910.00062

Nash, J. 1951, Annals of Mathematics, 54, 286

Newell, A. 1982, Artificial Intelligence, 18, 87

Nun, I., Protopapas, P., Sim, B., & Chen, W. 2016, AJ, 152, 71

Odena, A. 2016, arXiv e-prints, arXiv:1606.01583

Pacini, F. 1967, Nature, 216, 567

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. 2019, in Advances in Neural Information Processing Systems 32 (Curran Associates, Inc.), 8024–8035

Patle, A. & Chouhan, D. S. 2013, in 2013 International Conference on Advances in Technology and Engineering (ICATE), 1–9

Petrova, O. 2020, Semi-Supervised Learning with GANs: a Tale of Cats and Dogs, `https://blog.scaleway.com/semi-supervised/`

Quinlan, J. R. 1986, Mach. Learn., 1, 81

—. 1993, C4.5: Programs for Machine Learning (Morgan Kaufmann)

Radford, A., Metz, L., & Chintala, S. 2015, arXiv e-prints, arXiv:1511.06434

Radford, A. & Narasimhan, K. 2018, in Improving Language Understanding by Generative Pre-Training

Ransom, S. 2011, PRESTO: PulsaR Exploration and Search TOolkit

Rea, N. & Pons, J. A. 2015, Pulsar science with the SKA

Roberts, N., Lorimer, D., Kramer, M., Ellis, R., Huchra, J., Kahn, S., Rieke, G., & Stetson, P. 2005, Handbook of Pulsar Astronomy, Cambridge Observing Handbooks for Research Astronomers (Cambridge University Press)

Ruder, S. 2016, arXiv e-prints, arXiv:1609.04747

Ruderman, M., Zhu, T., & Chen, K. 1998, ApJ, 492, 267

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

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. 2016, arXiv e-prints, arXiv:1606.03498

Schapire, R. E. 1990, Mach. Learn., 5, 197

Schmidhuber, J. 2015, Neural Networks, 61, 85

Sermanet, P., Kavukcuoglu, K., Chintala, S., & LeCun, Y. 2012, arXiv e-prints, arXiv:1212.0142

Shapiro, S. L. & Teukolsky, S. A. 1983, Black holes, white dwarfs, and neutron stars : the physics of compact objects

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. 2014, J. Mach. Learn. Res., 15, 1929–1958

Sun, W., Li, C., & Ren, R. 2019, in Journal of Physics Conference Series, Vol. 1237, Journal of Physics Conference Series, 022142

Susskind, J., Anderson, A., & Hinton, G. E. 2010, The Toronto face dataset. Technical Report UTMLTR 2010-001, U. Toronto.

Tan, C. M., Lyon, R. J., Stappers, B. W., Cooper, S., Hessels, J. W. T., Kondratiev, V. I., Michilli, D., & Sanidas, S. 2018, MNRAS, 474, 4571

Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. 2013, in Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13 (New York, NY, USA: Association for Computing Machinery), 847–855

Thornton, D. 2013, PhD thesis, University of Manchester

Vapnik, V. N. 2000, The Nature of Statistical Learning Theory, Second Edition, Statistics for Engineering and Information Science (Springer)

Vega, R., Gorji, P., Zhang, Z., Qin, X., Rakkunedeth Hareendranathan, A., Kapur, J., Jaremko, J. L., & Greiner, R. 2021, arXiv e-prints, arXiv:2102.06164

Verbiest, J. P. W., Bailes, M., Coles, W. A., Hobbs, G. B., Van Straten, W., Champion, D. J., Jenet, F. A., Manchester, R. N., Bhat, N. D. R., Sarkissian, J. M., Yardley, D., Burke-Spolaor, S., Hotan, A. W., & You, X. P. 2009, Monthly Notices of the Royal Astronomical Society, 400, 951

Wang, P. 2020, in 2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL), 218–221

Wang, Y. & Ni, X. S. 2019, arXiv e-prints, arXiv:1901.08433

Wang, Y., Pan, Z., Zheng, J., Qian, L., & Li, M. 2019a, Ap&SS, 364, 139

Wang, Z., She, Q., & Ward, T. E. 2019b, arXiv e-prints, arXiv:1906.01529

Wilson, D. L. 1972, IEEE Transactions on Systems, Man, and Cybernetics, SMC-2, 408

Wolpert, D. H. 1992, in Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992], ed. S. J. Hanson, J. D. Cowan, & C. L. Giles (Morgan Kaufmann), 539–546

Woltjer, L. 1964, ApJ, 140, 1309

Yuen, B., Hoang, M. T., Dong, X., & Lu, T. 2020, arXiv e-prints, arXiv:2011.03842

Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., & Metaxas, D. 2016, arXiv e-prints, arXiv:1612.03242

Zhao, D., Chen, Y., & Lv, L. 2017, IEEE Transactions on Cognitive and Developmental Systems, 9, 356

Zhao, Z.-Q., Zheng, P., Xu, S.-t., & Wu, X. 2018, arXiv e-prints, arXiv:1807.05511

Zhou, Z. 2012, Ensemble Methods: Foundations and Algorithms, CHAPMAN & HALL/CRC MACHINE LEA (Taylor & Francis)

Zhu, W. W., Berndsen, A., Madsen, E. C., Tan, M., Stairs, I. H., Brazier, A., Lazarus, P., Lynch, R., Scholz, P., Stovall, K., Ransom, S. M., et al. 2014, ApJ, 781, 117

Zhu, X. & Ghahramani, Z. 2002, Learning from Labeled and Unlabeled Data with Label Propagation, Tech. rep.