Radboud University
Faculty of Science

Bachelor Thesis

Machine Learning in dedicated Supersymmetric Dark Matter searches at the LHC

Jasper van der Kolk

With supervision of
-
Prof. Dr. Wim Beenakker
Melissa van Beekveld (MSc)
-
Department of High Energy Physics

September 2016-August 2017

Contents

1 Acknowledgements

The one thing that amazed me every time I get in contact with scientific research is how much time people are willing to spend helping others. I certainly took up quite some time of many different people during my project. I would really like to thank these people for all their advice and explanations.

I must of course start with my supervisors, Wim Beenakker and Melissa van Beekveld. I think that the first real impression Wim got of me was when I walked into his room with an a4 filled with about 30 questions about the theory in Griffiths' *An introduction to particle physics*. Him walking me through all of them, answering them and explaining some extra background information is indicative of how I experienced Wim during my entire project. Always ready to help, taking his time and making sure I not only understood my project but also the framework it sits in.

My project was in fact an extension of Melissa's master project, and therefore I mostly worked together with her. Together, we tackled quite some problems, mainly involving the production of data. Melissa and Wim were a perfect team in supervising me, both having similar views and ideas, both taking their time to discuss the problems that came up during my project.

Then there are my machine learning guru's in Nijmegen, Bob Stienen and Luc Hendriks. They both gave me private lectures on machine learning which were extremely useful, giving me a head start on being able to implement several different tools. Besides this, they were always open for questions, via mail or in one of the many dark matter meetings.

The dark matter meetings were weekly events where everyone working on dark matter came together to discuss their results and difficulties. For me these could be quite useful, to get new input and keep everyone up to date. Mostly these were led by Sascha Caron, who reminded me often of the need to produce usable results and always gave me the feeling that I could really contribute to science, even during a bachelor project.

Sascha also helped me with finding a suitable foreign component of my project by getting me in contact with Daniel Whiteson of the University of California Irvine. I spent three great weeks there, learning a lot under the supervision of Daniel and Lars Hertel.

Finally I would like to thank the Honours programme for giving me the opportunity to do an extended bachelor project and making it possible to do part of my project at Irvine, which was a very valuable experience.

2 Introduction

At the Large Hadron Collider (LHC) at CERN in Geneva, particles are collided at immense energies and frequencies, to probe a world of physics that was previously inaccessible. Each collision produces large amounts of particles, many of which we already know. But there are many theoretical models that predict new particles, that should also be produced in the LHC. The problem is finding them. These particles decay very quickly, and are not detected directly. Their decay products are what is measured in the detector. How can we know which particles come from known particles (background) and which originate from new, undiscovered physical processes (signal)? In this thesis, we will look into the task of distinguishing signal from background. This will be done in two ways, first by contributing to an already existing strategy, and second by constructing a search strategy from scratch.

Of course, the term 'new particles' is quite vague. The new physics we will be looking into comes from the framework of supersymmetry, an extension of the standard model which describes all the particles we have observed so far. The interesting thing about supersymmetry is that it provides an explanation for dark matter. Astronomical observations indicate that regular matter, which makes up everything we can encouter in our everyday life, cannot be all there is. In fact, a large portion of the matter in the universe could be comprised of some unknown matter, which we call dark matter, because it does not emit light. In the first half of the theory section (sec. 3.1 - 3.4), we look into the concepts of supersymmetry and dark matter. These are both very complex subjects and we will therefore only go into the theory that is relevant for this thesis. We also shortly go into the LHC and one of the experiments that actually measures the particles, ATLAS (sec. 3.5).

In the second half of the theory section(sec. 3.7), we will discuss the method of how we are going to distinguish background from signal. Our strategy will be based on machine learning tools. Machine learning uses computer algorithms to find patterns in data. By feeding these tools computer generated data from collision events, patterns can be found that help distinguish between different kinds of data.

As we have said before, this project will be split into two subprojects. The same can be said about this thesis. The methods, results and discussion will be separated for each subproject. We will explain what algorithms were used, what the results were and what this means for search strategies at the LHC.

3 Theory

3.1 Dark Matter

One of the biggest unanswered questions in physics is what the universe is made of. We now believe that only about 5% of the energy budget of the universe is baryonic matter, the matter we know from day to day life. A far larger part is dark matter (DM), which makes up about 25% [1]. There are many theories as to what DM is, and this research is ultimately aimed at helping to answer this question. Before we go into all that, there is an important question we need to ask ourselves. Why do we believe that there is so much unexplained matter in the universe? This is because there are many astronomical observations that point in this direction. Here, we will discuss three of them.

3.1.1 Rotation Curves

The first of these observations is the rotation curves of galaxies. Galaxies, like our own Milky Way, rotate around their center. The speed with which objects in the galaxy rotate around their center depends on how far that object is located away from the center. Figure 1 shows an example of the measured rotation curve of galaxy NGC6503 [2].

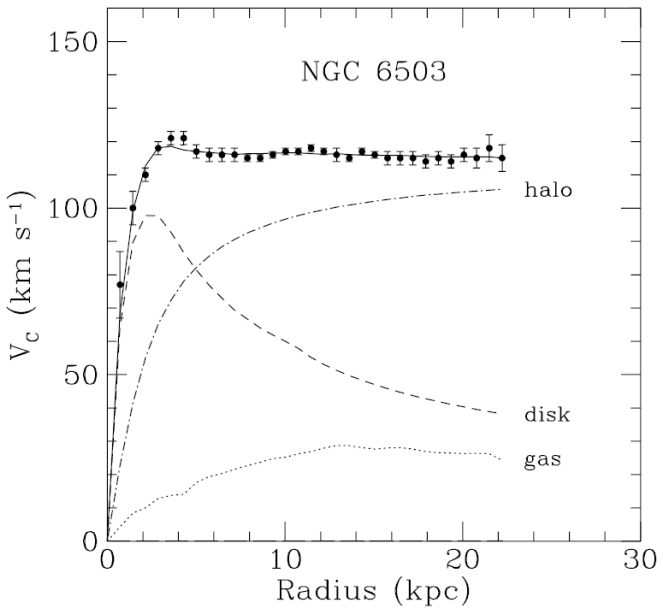


Figure 1: *The rotation curve of galaxy NGC6503 [2]. The dotted, dashed and dashed dotted lines are the theoretical contributions from gas, disk and halo respectively. We can clearly see that contributions from disk and gas (the 'normal' matter), do not give the observed curve. Only when a DM halo is added does the theoretical prediction match the measurements.*

Figure 1 shows us that at large radii, beyond the edge of the galactic disc, a flat behaviour of the rotation curve occurs. This means that over a large range of distances from the center of the galaxy, objects travel at the same speed. This is unexpected because we know from Newtonian dynamics that the speed of an object in orbit around a massive object should follow relation 1. It must be noted that this relation is only valid for spherically symmetrical galaxies. However, similar arguments can be made for non-spherical, more realistic mass density profiles of galaxies.

$$v(r) = \sqrt{\frac{GM(r)}{r}} \quad (1)$$

$$\text{with } M(r) = \int \rho(r) r^2 \mathrm{d}r \quad (2)$$

Here, G is Newtons's constant. We observe that the mass profile, $M(r)$, of *baryonic mass* decreases $\propto 1/\sqrt{r}$ at large r. However, we see that at large r, v is constant. This means that the mass density profile $\rho \propto 1/r^2$. Thus there must be something that gives rise to this deviating distribution [3].

Such observervations can be explained in different ways. A simple explanation is the presence of extra mass, which is altering the mass density profile. This mass could be in the form of non-luminous astrophysical objects, but also in the form of electrically neutral particles introduced by new physics. In this research, the mass is assumed to originate from such an unknown particle. However, you could also argue that the laws of gravity we know might be wrong or incomplete at large distances. For our example this would mean that although the mass distribution decreases with $1/\sqrt{r}$, some unknown gravitational effect causes the speed to become constant at larger r.

3.1.2 Bullet Cluster

So we have seen that there are two important explanations for the gravitational effects observed. Why then do we choose to investigate the existence of an unknown, new, elusive particle when it is not even clear if such a particle is necessary? Well, this is because there are also some important observations that cannot be explained by modified gravitational laws. One of these observations is the bullet cluster [1]. This is a cluster of galaxies that has been formed by two smaller clusters colliding. From the observational technique of gravitational lensing the mass distribution of this cluster has been deduced. Gravitational lensing is an effect described by General Relativity. Because mass curves space-time it changes the trajectories of particles, such as photons, causing mass to bend light around it. This can be used in astronomy to measure mass distributions, by looking at light from bright objects behind the massive object under investigation. This is very useful because it allows to not only observe mass that emits light, but also matter that does not, such as DM.

Figure 2: *Image of the bullet cluster, two colliding clusters. A combination of gravitational lensing and optical measurements have been used to identify DM (purple) and baryonic matter (pink).*

In figure 2 we can see four regions, two pink and two purple. The pink regions are made of baryonic gas, slowed down by friction upon impact. This gas is hot and emits photons, and can be observed by optical measurements. The purple areas have slowed down much less than the gas and have thus accumulated near the edges of the new cluster. Because the mass fraction of gas in clusters is much higher than that of objects like stars [4], we would expect that most of the mass would be near the center of the new cluster, because that is where the gas is located. However, gravitational lensing observations tell us that in fact most of the matter is located near the edges of the new cluster, in the purple areas.

This is a strong indication that there are unknown objects/particles that interact at most weakly (or else friction would have slowed them down). This has much less to do with gravitational effects, and thus cannot be explained with modified gravity.

3.1.3 CMB

A distinction can be made between cold DM and hot DM. The distinction is based on speed. Hot DM moves relativistically, cold DM does not. Most evidence points to DM being cold. One of the observations in support of this is the cosmic microwave background (CMB). This is the light that is left over from right after the universe became optically transparent, so a few seconds after the Big-Bang. An image of the distribution of this light can be seen in figure 3.

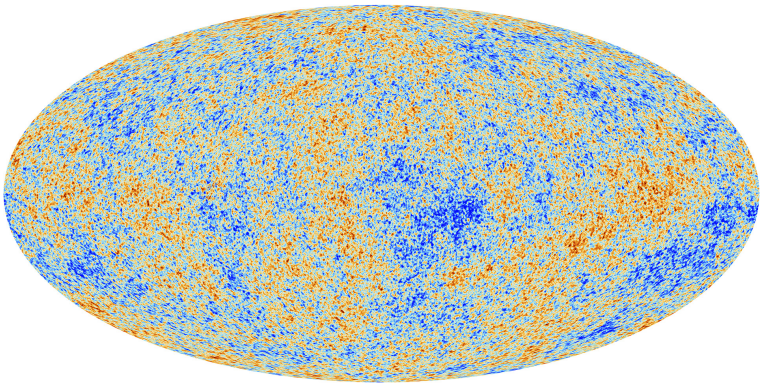


Figure 3: *Image of the CMB as observed by the Planck telescope. The colours indicate the intensity fluctuations of the light.*

The intensity is not the same across the image. We believe that it is DM that triggered this structure formation. The details of this are not relevant, but the idea is that tiny variations in the initial mass distribution would have increased due to mass (including DM) accumulating around areas of higher mass density. This is a positive feedback loop, which eventually leads to the large scale structures we see today. However, for this to occur, DM must be cold. If it was too hot, it would never accumulate near areas with more mass, never initiating the positive feedback loop.

3.1.4 The WIMP miracle

The following section has been based on ref. [5]. Both temperature T and DM mass m_χ have units of energy.

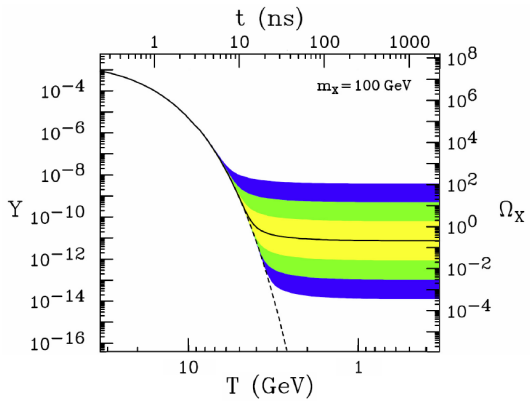
Now that we have made it plausible that we need a new type of particle to explain DM, an obvious question arises. What should this particle be? One of the most important candidates for particle DM is the class of WIMPs, the weakly interacting massive particles. These WIMPs are particles with masses at the weak scale, which is in the 10 GeV - 1 TeV regime. They are a candidate because they introduce something called the *WIMP miracle*. This is the fact that stable WIMP DM could give rise to the DM *relic density* (the amount of DM left in the universe). Several measures of the relic density will be used; Y and Ω_X. Introducing their exact definition requires information that goes beyond the scope of this research. Giving exact values of these quantities would thus also be less useful. For the arguments made in this section, it is enough to recognize them as measures of the density of dark matter left in our universe at the present moment. This section is aimed at introducing the concept of WIMPs and making plausible that they could be good DM candidates.

To understand how WIMPs could give rise to the correct relic density, we need to go back to the beginning of the universe. After the Big Bang, the universe was a very hot place. When the temperature of the universe T is higher than the mass of the DM particle m_χ, the particles are in thermal equilibrium, meaning that as many particles are created as are annihilated. However, as T decreased, eventually becoming smaller than m_χ, the abundance of the particles decreases with $\sim (m_\chi T)^{3/2} e^{m_\chi/T}$. At the present moment, the temperature of the universe is very low, and we would expect the amount of DM particles to have dropped to zero. However, the universe is not only cooling down, it is also expanding. This causes a *freeze out*: at a certain moment the DM in the universe is so dilute, that the DM particles are too far away from each other to interact. This in turn causes the number of DM particles to approach a constant. We can describe this process using the Boltzmann Equation:

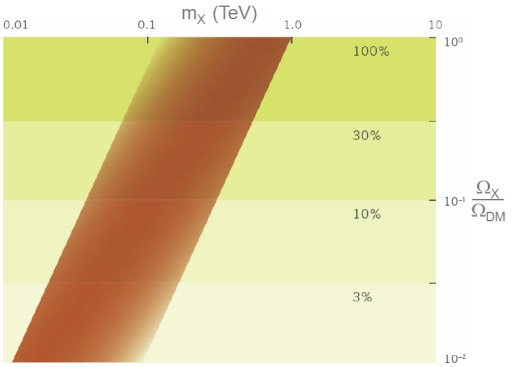
$$\frac{dn}{dt} = -3Hn - \langle \sigma_A v \rangle (n^2 - n_{eq}^2)$$

The first term on the right hand side describes the decrease of number density (n) due to dilution. Here, H is the Hubble parameter, which is a measure of the rate at which the universe expands. The second term gives the change of number density due to creation and annihilation. Here $\langle \sigma_A v \rangle$ is the thermally averaged DM annihilation cross section, which is a measure for the amount of events, i.e. how often a certain reaction occurs. The n^2-term comes from the DM annihilation reaction $XX \rightarrow SM$, with SM representing the Standard Model (non-DM) particles, and n_{eq}^2-term comes from the reverse, DM creation, process.

The relic density can be calculated by solving the Boltzmann equation numerically. However, this is a quite difficult procedure, and goes beyond the scope of this thesis. We will only discuss the results. This will be done on the basis of two figures.



(a) *The evolution of the density of a 100 GeV WIMP, represented by Y and Ω_X as a function of time after the Big Bang (t) and temperature (T). The solid line indicates the density evolution that leads to the correct relic density. The coloured areas show the evolution of the density when the annihilation cross section of the WIMP is changed. Each colour boundary indicates a change in the cross section by a factor ten. The cross sections by a factor $10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}$ going from to bottom.*



(b) *This image shows what values for the mass of the WIMP (m_χ) could give rise to the correct relic density, given reasonable values of the cross section (the brown spread). The y-axis states the fraction of the total amount of DM that the WIMP would explain.*

Figure 4: *Figures taken from ref. [5].*

From figure 4a we can gather that a 100 GeV WIMP could indeed lead to the correct relic density, if it has the correct cross section. Variations in the cross section will lead to different relic densities. We saw this dependence in the Boltzmann equation. The evolution of the dashed line indicates the evolution of the density of a particle that stays in thermal equilibrium. As we had expected, this cannot lead to the correct relic density. Figure 4a shows us the evolution of a WIMP particle with a single mass. Figure 4b shows us the range of masses that could lead to a correct relic density. For example, a particle with the same annihilation cross section but a lighter mass as the particle displayed in the left figure would lead to a relic density that is too low. However, if its cross section was also a bit lower, it could again lead to the correct value. The brown shaded area in figure 4b indicates the particle masses that could give rise to the correct relic density if the correct, reasonable cross section is chosen. We see that for 100% WIMP DM we get a mass range of 100-1000 GeV, very natural values for a weakly interacting particle. This is what is referred to as the WIMP miracle: a weakly interacting massive particle could very naturally lead to a correct relic density. Of course, not all DM has to be explained by a single particle. If we accept this, more masses and annihilation cross sections are acceptable. The supersymmetric DM candidate that will be introduced later falls into this category.

3.2 Brief overview of the Standard Model

Now we have convinced ourselves that particle DM is necessary and that the WIMP is a plausible candidate, we need to see if we can find a good WIMP candidate. The obvious place to start looking is the Standard Model of particle physics (SM). To do this, it is a good idea to first recap what we know about the SM. This section is entirely based on several chapters from ref. [6].

The SM of elementary particle physics describes the smallest building blocks of nature and their interactions. These particles can be grouped by several different properties. The first is spin. Particles with half integer spin are called *fermions*, whereas particles with integer spin are called *bosons*. Fermions can be divided into different subgroups. If the particle has a colour charge, and thus interacts via the strong force, it is called a quark. If it does not, it is a lepton. These groups can both be divided into three subgroups, called *generations*. There are twelve fermions. Six *flavours* of leptons distributed over three generations, and six *flavours* of quarks, also divided in three generations. The complete list of fermions is shown below:

Table 1: The leptons of the SM.

Generation	Flavour	Charge	Spin
First	e (electron)	-1	$\frac{1}{2}$
	ν_e (electron neutrino)	0	$\frac{1}{2}$
Second	μ (muon)	-1	$\frac{1}{2}$
	ν_μ (muon neutrino)	0	$\frac{1}{2}$
Third	τ (tauon)	-1	$\frac{1}{2}$
	ν_τ (tauon neutrino)	0	$\frac{1}{2}$

Table 2: The quarks of the SM.

Generation	Flavour	Charge	Spin
First	d (down)	$-\frac{1}{3}$	$\frac{1}{2}$
	u (up)	$\frac{2}{3}$	$\frac{1}{2}$
Second	s (strange)	$-\frac{1}{3}$	$\frac{1}{2}$
	c (charm)	$\frac{2}{3}$	$\frac{1}{2}$
Third	b (bottom)	$-\frac{1}{3}$	$\frac{1}{2}$
	t (top)	$\frac{2}{3}$	$\frac{1}{2}$

Before we look at the bosons, we first need to introduce the four fundamental forces of nature. These are, in decreasing strength: strong, electromagnetic, weak and gravitational forces. Gravity cannot be treated in the same quantum mechanical way as the other three forces and will thus not be treated here. All forces need to be considered on a quantum scale to be applicable in elementary particle physics. In this context we couple a force to a mediator. These mediators transfer a specific force between particles that *couple* to this force.

The first force we will discuss is the electromagnetic force. This is the same force that we know from the classical theory of electromagnetism. However, on the quantum scale we need to look at the quantum version, called quantum electrodynamics (QED). The electromagnetic force works on particles which have charge and is mediated by massless photons.

The second elementary force we will discuss is the strong force. Because the strong force works on very tiny length scales, it does not have a classical equivalent. The quantum mechanical theory of the strong force is called QCD, quantum chromodynamics. We immediately recognize the Greek word for colour in that name. Colour is an intrinsic property of particles that couple to the strong force, which in the SM are quarks and *gluons*. There are three types of quark colour: red, green and blue. Quarks carry colour, antiquarks carry anticolour: anti-red, anti-green and anti-blue. We should stress that quarks don't actually have a physical colour, it is a name given to one of the quark's quantum numbers. Like the photons in QED, the force mediator in QCD are the massless gluons. Gluons are bi-coloured, meaning they have one color and one anti colour. All measurable particles in nature are colourless or white, so it is impossible to find a free quark or gluon. This is called *confinement*. They always combine to form a *baryon* (like the proton) or a *meson* (like the pion). Baryons are made up of three quarks in a colourless combination. Mesons are made of a quark and an antiquark, also in a colourless combination. Together the mesons and baryons are called *hadrons*.

The final force we will talk about is the weak force. It does not have an abbreviation like the other two forces do. Another thing that sets this force apart from the other two is the fact that the mediators have mass. These couple to all leptons as well as quarks. The weak force has three massive mediators, the Z-boson, which mediates the neutral interaction, and the $W^{\pm}$-bosons, which mediate the charged interaction. These mediators, as well as the photon and the gluon all have spin 1. Together they are known as the gauge bosons. Finally we have the Higgs mechanism, which is linked to mass generation. This mechanism introduces the final particle in the SM, the Higgs boson. It has spin 0 and is massive.

3.2.1 Mixing

In the previous sections, we omitted one complication from our discussion on the weak force. Observations indicate that it is not the mass-eigenstate quarks (up,down etc.) that the weak force couples to. There seems to be some mixing between the different generations. This means that the weak eigenstates of the quarks are not (u,d), (c,s), (t,b) but (u,d'), (c,s'), (t,b') where d',s' and b' are defined as follows.

$$\begin{pmatrix} d' \\ s' \\ b' \end{pmatrix} = \begin{pmatrix} V_{ud} & V_{us} & V_{ub} \\ V_{cd} & V_{cs} & V_{cb} \\ V_{td} & V_{ts} & V_{tb} \end{pmatrix} \begin{pmatrix} d \\ s \\ b \end{pmatrix}$$

The matrix in the middle is called the CKM-matrix, and it tells us how the different mass eigenstates mix in the charged interaction.

3.2.2 Electroweak force

One other thing to take into account is that the electromagnetic force and the weak force are actually not two distinct forces, but can be regarded as manifestations of the electroweak force, which is mediated by its own set of gauge bosons. These electroweak eigenstates are the B, W_1, W_2, W_3 bosons. These mix, just like the quarks in weak theory, to create the mass eigenstates we recognize:

$$\begin{aligned} \gamma &= \cos(\theta_W)B + \sin(\theta_W)W_3 \\ Z &= -\sin(\theta_W)B + \cos(\theta_W)W_3 \\ W^{\pm} &= \frac{1}{\sqrt{2}}(W_1 \mp iW_2) \end{aligned}$$

Here θ_W is the electroweak mixing angle.

3.2.3 Possible WIMP candidates

Now we have introduced all relevant parts of the SM, we can return to our original question and see if we can find a good WIMP candidate. It turns out that we cannot. One of the properties of the WIMP is that is must be solely weakly interacting. This means we can disregard all quarks and charged leptons. This leaves us with only the neutrinos and the the Z and Higgs boson. The latter two aren't good candidates because they are very unstable. Neutrinos would be our best shot, but sadly their mass is too small to produce the correct relic density. They are also relativistic, meaning they belong to hot DM, and we are searching for a cold DM candidate. So, we need to look somewhere else for our DM candidate.

3.3 Supersymmetry

The SM is not a complete description of nature. To fix its shortcomings, theorists come up with extensions to the SM. One of these extensions is supersymmetry (SUSY). SUSY says that every SM particle should have a partner, which has exactly the same quantum numbers except for its spin. The spin changes by 1/2, so bosons become fermions and vice versa. However, this would also mean that the mass of the partners would be the same. If this were the case, we should have already observed superpartners. The case is that SUSY is a broken symmetry, which means besides having different spins, the superpartners also have a different mass. It needs to be said that there are many different SUSY breaking models. The generic SUSY model we will be looking at is the Minimal Supersymmetric Standard Model (MSSM), which includes all SUSY breaking models with a minimal amount of particles.

So each particle has its own superpartner. The partners of the quarks and leptons are called squarks and sleptons and have spin 0. The names of the partners of the SM fermions are all constructed by adding the prefix s. The superpartners of the SM gauge bosons are called gauginos and have spin 1/2. The suffix -ino is added to the names of the SM gauge bosons. It is important to note that the electroweak gauginos are superpartners of the electroweak eigenstate gauge bosons, so W_1, W_2, W_3, B. Finally there is the Higgs boson. The MSSM requires five higgs states, two charged and three neutral. The Higgs superpartners also have spin 1/2 and are again named by adding the suffix -ino. The symbols of all superpartners are constructed by adding a tilde to the SM symbol (e.g. $\mu^{\pm} \rightarrow \widetilde{\mu}^{\pm}$).

Just like in the SM, the mass eigenstates of the gauginos are not the same as the electroweak eigenstates. Mixing again occurs, to create the *neutralinos* and the *charginos*. The neutralinos are mixed states of the neutral electroweak gauginos, belonging to B and W_3, and the two neutral higgsinos. This creates four new particles, which are all neutral and are ordered by increasing mass. The charginos are mixed states of the corresponding charged gaugino, so $\widetilde{W}^{\pm}$, and corresponding charged higgsino. These again are ordered by increasing mass. Eventually, we have 4 neutralinos and 4 charginos:

$$\begin{aligned} \textit{neutralinos}&: \widetilde{\chi}_1^0,\ \widetilde{\chi}_2^0,\ \widetilde{\chi}_3^0,\ \widetilde{\chi}_4^0 \\ \textit{charginos}&: \widetilde{\chi}_1^{\pm},\ \widetilde{\chi}_2^{\pm} \end{aligned}$$

Different MSSM models result in different neutralino and chargino compositions. If a mixed particle is composed of mostly Bino ($\widetilde{B}$), it is called Bino-like. The same goes for Higgsino and Wino ($\widetilde{W}_{1,2,3}$).

The last thing we need to introduce about the MSSM is R-parity conservation. The most important thing to note is that supersymmetrical particles have R-parity -1 and each SM particle (including all Higgs states) has R-parity +1. If interactions conserve R-parity, supersymmetrical particles can't decay into (only) SM particles. This in turn means the lightest supersymmetrical particle (LSP) is stable!

We have introduced a lot of new concepts. However, we should remember our goal: to find a good WIMP candidate. Could one of the SUSY superpartners be suitable? It turns out this is indeed the case and that the lightest neutralino is a great WIMP candidate. Why is this? Well, first of all it is charge and colour neutral, so only interacts weakly. Second of all, it has mass so it could explain the gravitation effects observed by astronomers. Its mass can also be in a range that allows it to produce the correct relic density. And finally, in many SUSY breaking models $\widetilde{\chi}_1^0$ is also the LSP, and therefore stable!

3.4 Galactic center excess

So far we have seen that the lightest neutralino of the supersymmetric model could be a good candidate for DM. However, there are many parameters like masses and couplings in the MSSM. Which models should we look for? To answer this question, we need experimental input, in this case in the form of the photon excess coming from the center of the Milky Way.

As we have stated before, there is a lot of DM in our galaxy, especially near the center. This is because of gravitational effects: because there is a lot of baryonic matter near the center, DM is pulled towards it. This in turn pulls more baryonic matter towards the center. Via annihilation, this DM could produce photons (indirectly). It has been shown that certain DM models can explain the photon excess in the 1 GeV $\lesssim E_\gamma \lesssim$ 5 GeV range measured by the Fermi-LAT telescope via these annihilation photons. Observational data was fitted using SUSY parameter scans to find which combination of parameters would lead to the best fit to the excess amount of photons coming from the Galactic center [7]. For our purposes the most important parameters to look at are the masses of the lightest neutralino and the lightest chargino. Figure 5 shows which model points would explain the excess best (a lower p-value means a better fit). We can see that the difference in mass between these two particles, the so called mass gap ΔM, can be quite small (0 GeV $\lesssim \Delta M \lesssim$ 90 GeV). Why this is an interesting observation will become clear in the next section, where we will explore the direct search for these DM candidates.

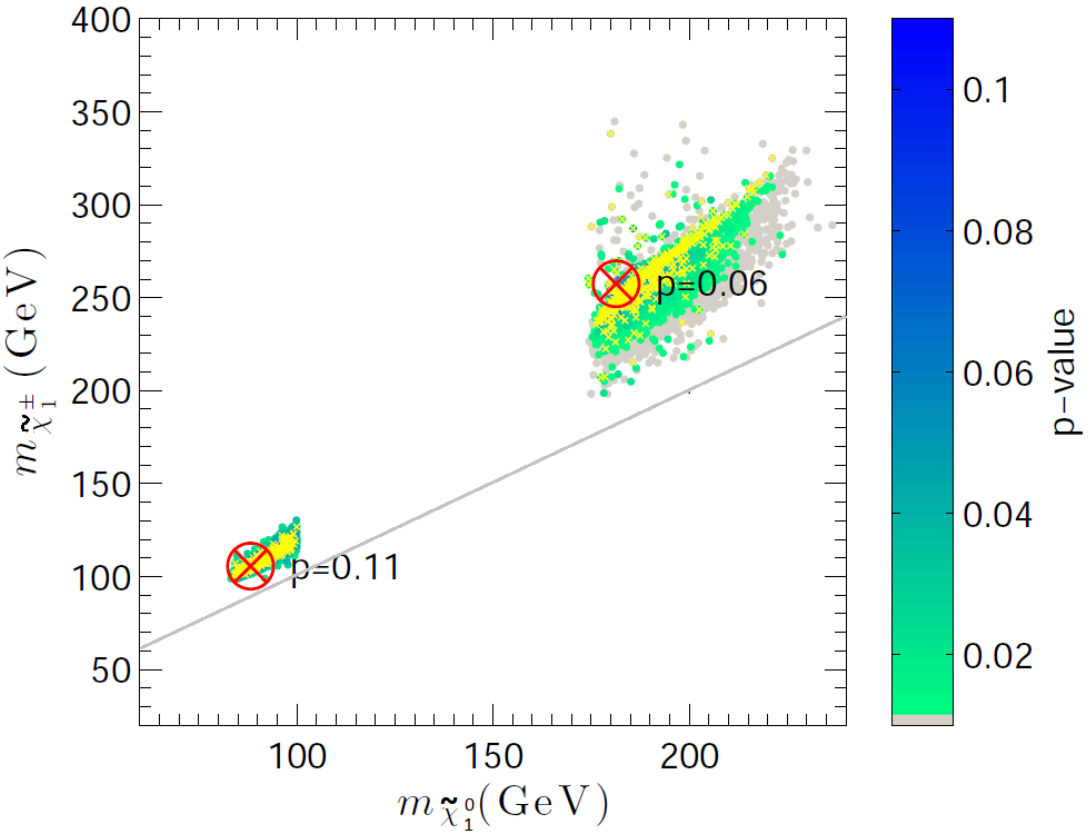


Figure 5: *The best fit SUSY models that could explain the Galactic center excess (GCE). A lower p-value means a better fit. The yellow overlay indicates those models that result in a correct relic density. The grey line indicates the models where the mass gap is zero:* $m_{\tilde{\chi}_1^\pm} = m_{\tilde{\chi}_1^0}$. *Adapted from ref. [8].*

3.5 Collider searches

There are many different ways of looking for DM, but the way we will look into involves particle colliders, in particular that of the Large Hadron Collider (LHC). In this collider, protons are accelerated and collided into one another, creating many different particles. Every collision is called an *event*. If SUSY exists and it includes particles with masses in the range accessible to the LHC, these particles could be created in these collisions. If this is the case, we should be able to find signs of their creation. And if we do not find signs of their creation, we should be able to exclude certain mass and cross section ranges of the particles. This is why SUSY should be a broken symmetry; SUSY particles with low masses have all been excluded because no sign of them has ever been found. These signs would be found in one of the detectors at the LHC. The LHC accelerates the protons and collides them and the detectors measure everything that comes out of those collisions. There are a few detectors, each with a different focus. The general-purpose detector we will be focusing on is ATLAS.

3.5.1 ATLAS

ATLAS is one of the four detector at the LHC. It has several layers, each designed to measure a specific particle. The precise details of ATLAS are irrelevant to this research. It is however important to know what it can measure. At first order we could say that it can measure particles and their four-momenta $(E, \vec{p})$. For example, it can detect the energy, momentum and recognise electrons. However, there are exceptions and complications.

One of these complications comes from quarks. As we have seen, quarks always combine to form baryons or mesons. However, they are not created as such. In the collision at the LHC, they are created as free quarks. They have very high energies, allowing them to very shortly travel as free quarks. There is so much energy that a quark-antiquark pair can be created. The antiquark pairs with free quarks to form a meson. The remaining quark now has a lower energy because some of the energy was used to create the quark-antiquark pair. However, there is still a free quark with a high energy. So the whole process is repeated several times until the energy of the remaining quark is so low that it can bind to a hadron. What we are left with is a group of hadrons all traveling in about the same direction. This cluster of hadrons is called a jet. ATLAS can measure the four-momenta of these jets.

The second important complication is the missing transverse momentum. In the LHC, the two protons are collided head on. This means that the total momentum transverse to the beam pipe is zero. Because of conservation of momentum the sum of all transverse momenta of all collision products (the particles created by the collision), should also be zero. However, as noted before, ATLAS cannot measure every type of particle. The most notable of these are the neutrinos. They only interact weakly and do not decay in the detector, which can therefore not detect them. This means that part of the transverse momentum is carried off by the neutrinos. The same would be true for a potential DM particle. It could at most interact weakly (and gravitationally) and would therefore fly right through the detector. This creates an unbalance in the transverse momentum. The momentum carried off by the undetected particles is missing, and is therefore called missing transverse momentum. This quantity can be calculated by adding up all the measured transverse momenta and seeing how much the balance is short of being zero. It should be noted that in the following we will be talking about missing transverse energy ($\not{E}_T$, MET). However, because energy and momentum can easily be related via the mass of the particle, this distinction is not very important.

To properly introduce the kinematic variables used in this research we first need to get a feeling of what coordinate system we will be using. We will be using spherical coordinates as can be seen in figure 6. It should be noted that instead of using θ, the coordinate pseudorapidity η is used, which is defined as follows:

$$\eta = -\ln\left(\tan\frac{\theta}{2}\right)$$

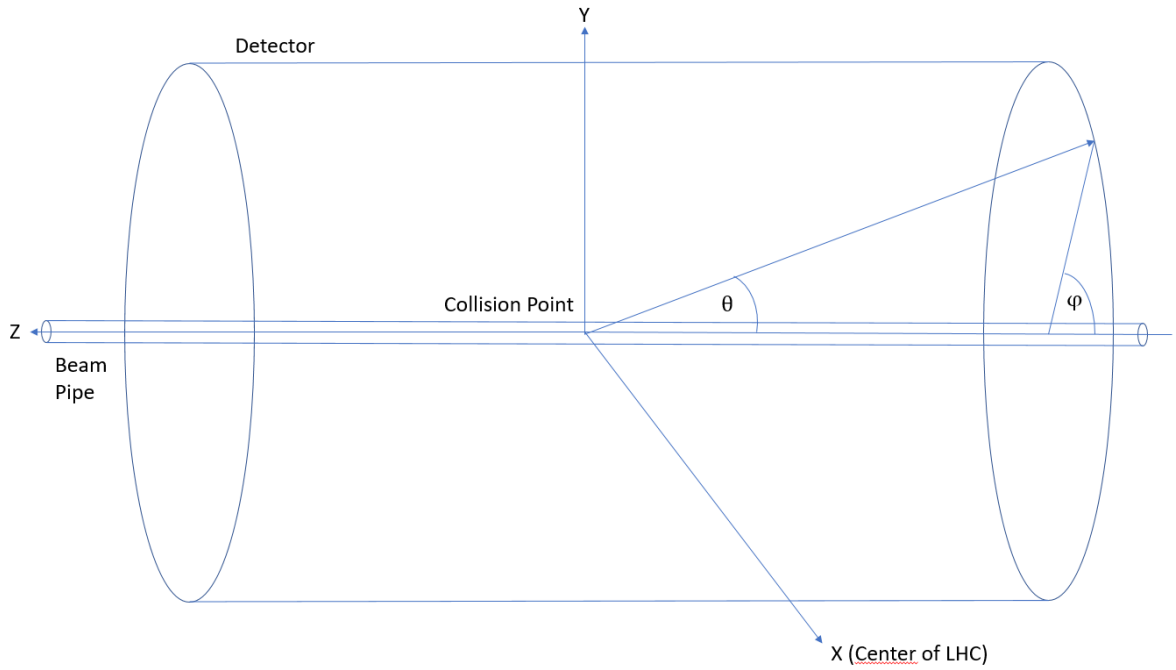


Figure 6: *The basic structure of the ATLAS detector, including the coordinate system it uses.*

Now we have seen in what kind of coordinate system we are operating, we can introduce some variables that will be used.

- $\Delta R = \sqrt{\Delta\phi^2 + \Delta\eta^2}$: A measure for the angular distance between two particles, with $\Delta\phi$ the difference in the polar coordinates and $\Delta\eta$ the difference in pseudorapidity.
- $M_{l+l-} = \sqrt{(E_{l+} + E_{l-})^2 - (\vec{p}_{l+} + \vec{p}_{l-})^2}$: The combined invariant mass of two particles with opposite charge (in this case leptons).
- $P_T = \sqrt{p_y^2 + p_x^2}$: The momentum of a particle perpendicular to the beampipe.

3.5.2 Signal - Background selection

Say that you are looking for a new particle that decays to a pair of electrons. ATLAS could certainly detect these electrons, but how could you know they were the result of some new particle instead of the many SM processes result in two electrons? The problem is that for every event where a SUSY particle is made, there are many events where only SM background processes occur. ATLAS selects the signal from the background by gathering all the events and applying cuts to different kinematic variables. We know what SM electrons should 'look' like (what properties they should have) and we know what electrons from new particles should look like. By finding the properties that distinguish signal from background, a lot of background can be cut away. For example, we could say that all events that have $\not{E}_T < 10$ GeV should be cut away, because the signal has a high $\not{E}_T$ signature. Eventually, you hope to be left with pure signal, i.e. events that only the existence of the new particle can explain. Now, a

pure signal can never be a reality, because signal and background often look very similar, and there are always uncertainties on the theoretical models that tell you the properties of signal and background events. ATLAS uses a significance measure to say how likely it is that a new particle has been found (or its mass excluded for that matter), called the Z_N-value. We will refer to this value as the significance. The Z_N value indicates how many standard deviations a new-physics model differs from the SM. Discovery is mostly claimed when the value exceeds 5σ, whereas a model is excluded if you expect to see more than a 3σ excess, but nothing is found.

3.5.3 Process

It is now time to introduce the specific process we will be looking at in this research. It can be seen in figure 7. Two quarks interact via the weak interaction, producing a W boson. This boson then creates a lightest chargino and a second lightest neutralino. Both of these particles decay via the weak interaction into the lightest neutralino (the WIMP candidate) and a weak gauge boson. These gauge bosons then decay into a pair of leptons. These final six particles enter the detector. As we have seen, the LSP and neutrino cannot be detected and contribute to the MET. The three charged leptons are detected by the ATLAS detector. This means this process has a specific signature: $3l + \not{E}_T$.

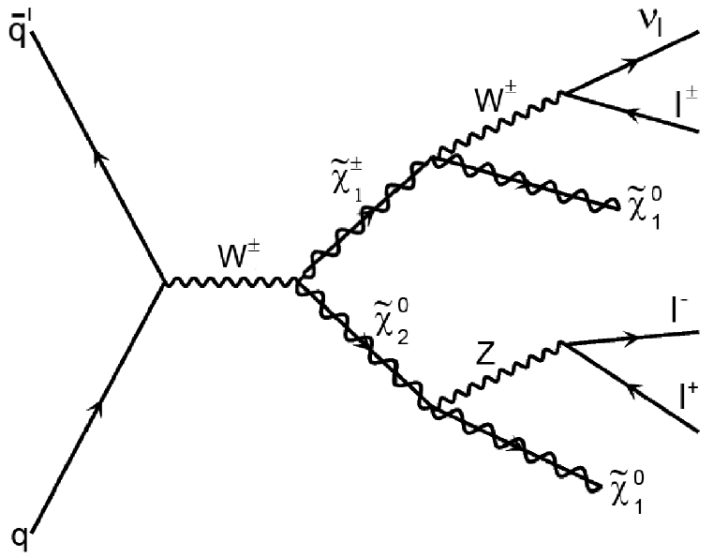


Figure 7: *The Feynman diagram of the process we will be investigating [9].*

An important property of this process is the presence of an opposite sign - same flavour (OSSF) charged lepton pair, which originates from the Z-boson.

3.5.4 Current searches

Several searches for the process introduced in the previous section have already been conducted or proposed. We will focus on the strategy by the ATLAS collaboration. The solid red and purple lines in figure 8 show the exclusion limits for the masses of the SUSY particles in the process shown in figure 7, which is the most important exclusion limit for our research. The purple limits belong to the LHC-run where the collisions had a center of mass energy of 8 TeV. The red limits are newer and belong to the LCH-run with a center of mass energy of 13 TeV. The lines with different colours belong to different processes. ATLAS unrealistically assumes that the *branching ratio* to two LSP's and a W and Z boson is 100%. This means

that all $\widetilde{\chi}_1^{\pm}$ and $\widetilde{\chi}_2^0$ particles decay into these products. For this search, a model is chosen where these SUSY particles have the same mass. Note that we will not be making these assumptions in our research. The most important line in the figure is the red one, which indicates the observed limit. Everything below that line can be excluded with 95% confidence level.

What we can see is that for compressed models, where $m_{\widetilde{\chi}_2^0} \approx m_{\widetilde{\chi}_1^0}$, the limits are not very strong, meaning that not a lot of models can be excluded. This is especially the case for the 13 TeV run. The general trend is that the larger the mass gap, the more models can be excluded.

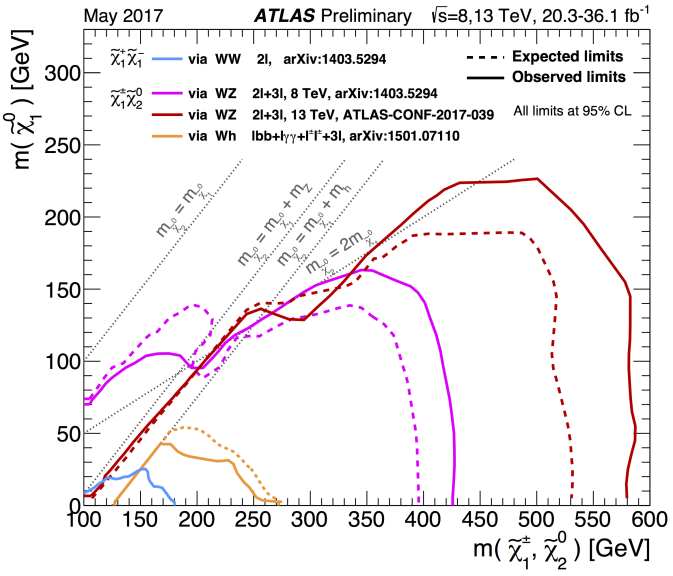


Figure 8: *The limits set by ATLAS on a specific SUSY model [10], where* $m_{\widetilde{\chi}_1^{\pm}} = m_{\widetilde{\chi}_2^0}$. *The red line indicates the limits as has been observed for the* $\sqrt{s} = 13$ *TeV run. Several lines indicate models where a certain relation between* $m_{\widetilde{\chi}_2^0}$ *and* $m_{\widetilde{\chi}_1^0}$ *apply.*

3.5.5 Search strategy for compressed spectra

We have seen that the limits placed by ATLAS are not as strong for compressed models. This is a problem, because these are the models that might be able to explain the GCE. We therefore need a way to become more sensitive to these types of models. We will look at a proposed search strategy introduced in ref. [9]. This paper stresses the importance of a search directly aimed at those compressed models.

There are several choices that the ATLAS collaboration makes that directly lead to their search strategy being less suitable in this mass region. First of all, the ATLAS strategy relies on triggers where $P_T(l) > 20$ GeV. This means that events are only saved as such if the transverse momentum of their most energetic leptons is higher than 20 GeV. However, for small ΔM models, this presents a problem. If we assume that the energy of the Z boson is spread evenly among its two daughter leptons, we can see that the momentum of each of these leptons is roughly bounded by $(m_{\widetilde{\chi}_2^0} - m_{\widetilde{\chi}_1^0})/2$. So we will have a problem if the mass difference drops below $40 - 50$ GeV.

Another problem that makes the ATLAS strategy less effective for compressed models is the fact that they look for large $\not{E}_T$. It is true that the LSPs carry off most of the energy of the next to lightest supersymmetric particles (NLSPs), in this case the $\widetilde{\chi}_1^{\pm}$ and $\widetilde{\chi}_2^0$. This is why it is logical to expect high $\not{E}_T$. However, this does not necessarily need to be the case. The

LSPs are often produced back to back. This allows for their momenta to be cancelled (to some extent) in the equation that is used to calculate the $\not{E}_T$.

Selecting the OSSF pair poses another challenge. Doing this correctly is important when reconstructing an event that gives information about the new physics. If all three leptons have the same flavour, there are two candidate OSSF pairs. One is correct, with both leptons coming from the Z-boson. The other is wrong, because one lepton originates from the $W^{\pm}$ boson. ATLAS selects the pair of leptons that has an invariant mass M_{l+l-} close to the Z-boson-mass. Because of conservation of momentum, the invariant mass before and after a decay must be the same, so the invariant mass of the lepton pair will be the same as that of the Z-boson. So selecting the pair based on the Z-boson-mass seems like a good idea. However, we are looking at compressed spectra, where the mass gap between the lightest and second lightest neutralino is smaller than the mass of the Z-boson. There is not enough energy available to produce the boson on shell, and so a virtual Z-boson is created. A virtual Z-boson can have a lower invariant mass, so the lepton pair produced in a compressed process will also have an invariant mass lower than the Z-mass. Ref. [9] proposes to select the OSSF-pair based on the minimum separation (smallest value of ΔR). Other possibilities include taking the pair with the smallest M_{l+l-} or with the highest P_T per lepton.

3.6 Research Question

We have now seen why we need an explanation for DM, why the LSP could be a good WIMP candidate and what attempts have already been made to find it. In this research we will attempt to improve on the current search strategies. To do this we will be using machine learning techniques. Machine learning uses computer algorithms to search for patterns in data, which allows it to distinguish between different types of data points. The details of machine learning will be explained later.

In this thesis we will be researching two different questions. The first arises from the difficulties of selecting the correct OSSF pair. The goal of the first part of this research is to see if machine learning techniques can be used to classify the final state leptons as belonging to the OSSF pair or not. This leads to the first research question:

How can machine learning tools be used to classify final state leptons as pair or non-pair leptons?

In the second part of this research we will attempt to use machine learning tools to create an entirely new strategy. Machine learning will be used to classify events instead of leptons. We want to see if patterns can be found that allow for the distinction between events belonging to background processes and events belonging to signal processes. This leads to the second research question:

How can machine learning tools be used to classify Monte Carlo simulated events as belonging to signal or background processes?

3.7 Machine Learning

3.7.1 The basics

We are now going to look a little deeper into machine learning. In machine learning, data is provided to an algorithm, which then finds patterns in that data. Using these patterns, it can then predict an output for new data points. This output is called a *target* or *label*. There are two types of machine learning: *classification* and *regression*. In regression the target is continuous. An example of this type of machine learning is fitting a function. Given a certain $x \in \mathbb{R}$ what would $y \in \mathbb{R}$ be? In that case, x is a data point and y is the target. In classification problems the target or label are discrete and they tell us to which class a point belongs. An example of this would be predicting if someone has a disease. The data point would be a patient and the target would be positive or negative, i.e. the patient does or does not have the disease. Because our data points belong to classes (pair or non-pair and signal or background) we will be looking into classification.

Another distinction can be made by whether or not a target is provided with the data point. If it is, the machine learning problem is said to be *supervised*. If it is not, it is a unsupervised problem. In a supervised problem, we give the algorithm a lot of data, providing each data point with a target. Each data point contains an *attribute set*. These are the specific properties of a given data point. In machine learning, these properties are called *features*. In the medical example these would be the patients symptoms. The algorithm then tries to find patterns in the data that predict the target. If we then provide new data without a target, it can predict what the target should be. This is the route we will be taking in this research. Our problem calls for supervised classification.

To make sure the previous part is clear, I will apply the terms mentioned above to an example. Figure 9 shows a classification problem. There are three classes, red, green and blue. The x and y axis represent two features and each point is a data point. This means that (x,y) form the attribute set of the data point. The colour of each point indicates its class, or target. The algorithm then tries to relate the features to a target. This is represented here by the coloured areas. If a data point is in the red area for example, the algorithm says it has class red. So if a new data point were to have (x,y) = (4,5) it would predict that it has class red. The boundaries between the areas are called *decision boundaries*. The plain in which all this happens is called the *feature space*, which has the same amount of dimensions as the number of features (so in this case 2).

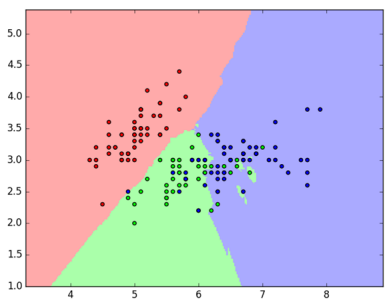


Figure 9: *Visualisation of machine learning. Each point in feature space represents a data point, each colour represents a class. The coloured areas indicate what the tool has learned [11].*

3.7.2 Model selection and overfitting

We have so far been very vague about the algorithms (we will also use the terms tools or classifiers) that find the pattern in the data. This is because there are many different algorithms that use different techniques to do their job. However, they all have something in common. They are all *tunable*. By altering them slightly, they can be improved. These improvements are made by adjusting the *hyperparameters* of the classification algorithm. The prefix hyper- is added to prevent confusion with the parameters of the data, the features. The patterns in the data are found by the algorithm itself, but the hyperparameters must be tuned by the user. The same goes for the size of the data sets used. For this quantity, the bigger the better applies. We will therefore take (arbitrarily) large data sets throughout the research.

This leads us to the next important concept in machine learning: *overfitting* [12]. The basic idea is that an algorithm can be made very complex by tuning its hyperparameters. An algorithm is complex when its decision boundary is complex. This can be made clear by again looking at figure 9. Most of the points are grouped together, but there are some outliers: some of the blue dots lie in the area that is dominated by green points and vice versa. Now we could tune the algorithm to make a very complex decision boundary, so that every data point of the training data lies in its corresponding area. However, it is very likely that when new data points are added for the algorithm to predict, these complex decision boundaries will not reflect reality, and some points will be classified incorrectly. Overfitting leads to very good classification of training data, but to poor classification of new data. The predictor is not *generalisable*.

To prevent this, we can separate our initial data, with which we create our predictor, into two groups: training and validation data. The training data is given to the algorithm to find the decision boundaries. The validation data is then used to tune the hyperparameters. This process is called *model selection*. In this context, a model is just an algorithm with a specific set of hyperparameters. Because the word 'model' can be confusing because of its use in the context of SUSY models, we will not use it in the context of machine learning. Checking a tool with the validation data will prevent overfitting, because tools which are overfitted will have poor performance on the validation data, and will thus be rejected. However, because you are using the validation data to tune the hyperparameters, you are still in danger of overfitting. When trying to get the best performance by tuning you could be tuning the hyperparameters in such a way that the predictor works well on the validation data but will still not be very generalisable. To make sure that this is not the case, it is often wise to split the data in three instead of two, with the third set being the test set. This is the final check to see how well your model works. The test set is used only once, after all tuning is complete, to assess the actual performance of the predictor.

Now we understand machine learning in general, we can look at it in more detail, and at the different algorithms specifically. In this thesis we will be looking at decision trees, random forests, support vector machines and neural networks. In the following sections, we will explain the basic ideas of these tools.

3.7.3 Decision Trees

Decision trees are one of the most simple classification tools. The basic idea is to take consecutive binary cuts in feature space until you are left with a decision about what class a data point belongs to. To clarify this, let's look at an example. The example has two features, x and y, and three classes, red, blue and yellow.

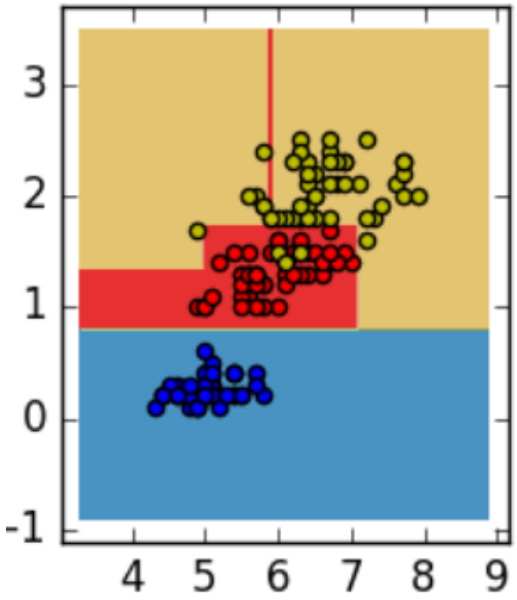


Figure 10: *Visualisation of ML by a decision tree. The lines between the different coloured areas have been learned by the tool to best separate the data [11].*

We can see that the decision tree has partitioned the feature space by drawing lines in it. For example, it has said that if $y < 0.9$, the class is blue. It has also found that if $0.9 < y < 1.4$ and $x < 7$ the class is red. By combining all these conditions it has divided up the feature space. We can also see an example of what could be overfitting. At $x \approx 6$ we see a sliver of red in the yellow area. This is probably caused by an outlier, and doesn't represent the truth.

So far it is not clear why this tool is called a decision tree. To understand this we will again look at figure 10, disregarding the sliver of red we previously mentioned. It is straightforward to see that this figure can also be represented as shown in figure 11.

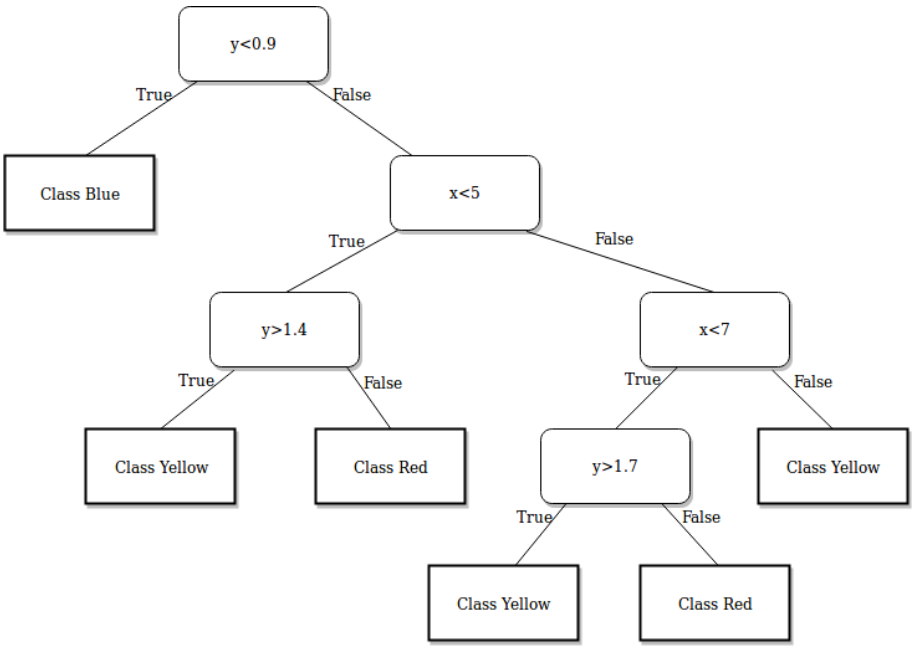


Figure 11: *Visual representation of a decision tree decision process. We start at the top and work towards the bottom, each end note being a classification.*

The decisions form a structure that looks like an inverted tree. The terminology of the architecture of such a tree stays with the tree metaphor. Each true/false line is called a *branch* and each final decision is called a *leaf*. The first condition is called the *root*.

Now we have introduced the basic idea and architecture of a decision tree we can look into how best to tune it. As we saw in figure 10, this type of tool is prone to overfitting. How can we counteract this? One way to do this is by setting a maximum depth to the tree. You can only have a specific amount of conditions before a decision has to be made. This prevents the creation of ever more complex decision boundaries.

One final thing we should introduce about decision trees before we continue to random forests, is feature importance. A decision tree algorithm can tell you how important a feature was when separating the data into classes. For example, consider a binary classification problem with two features, x and y. If all points of class A lie below y=3 and all points of class B lie above it, then the tool can classify all points by just using feature y. This feature is thus much more important than feature x.

3.7.4 Random Forests

As stated in the previous section, one of the biggest problems with decision tree algorithms is over-fitting. One of the solutions to this problem is using a random forest[13]. A random forest is basically a group of decision trees that all classify a certain point separately, and then vote over which it is. This process is called *bagging*. The best results are obtained if the trees are uncorrelated. However, they use the same features and try to describe the same system, so they are obviously not uncorrelated. The idea of a random forest is to decrease this correlation by training each tree with a selection m out of the total p features. A typical value for m is $\sqrt{p}$. However, this is not always the best choice. This could for example be the case if there are a lot of features but only a few relevant ones. The probability that a tree is then grown with relevant features is small. Here, m is a tuning parameter.

3.7.5 Support Vector Machine

One of the big problems with decision trees and random forests is that they can only draw straight lines through feature space, and are therefore mostly effective in the classification of linear problems, where the decision boundaries between classes are linear. Support vector machines (SVMs) use linear models to implement non linear class boundaries [14]. We will not give a full mathematical foundation of this, as it goes beyond the scope of this research. Instead, we will try and give an intuition for what this tool does, in stead of how it works exactly.

How does a SVM work? To understand this we first need to introduce the maximum margin hyperplane. To illustrate this concept we will first look at the linear, non-mapped 2D feature space, with binary data that is separable in that space. We can visualise this as follows:

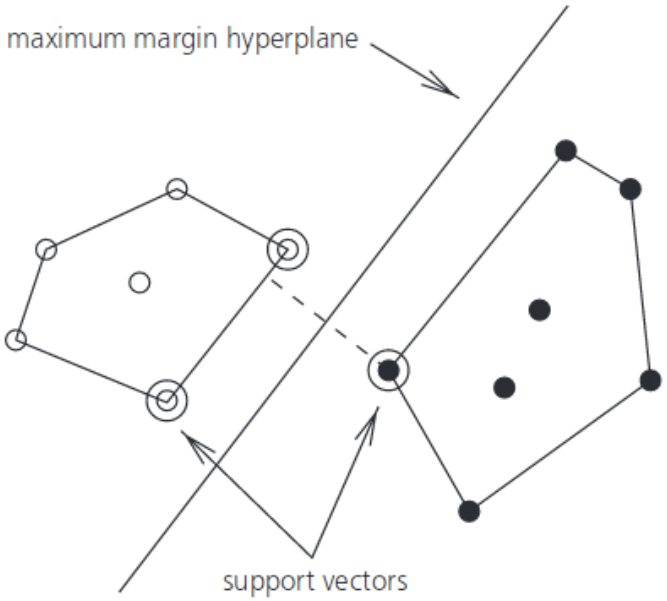


Figure 12: *The maximum margine hyperplane. This is the line that maximally separates the data points from two different classes, represented here as filled and empty dots. The points that lie closest to this plane are the support vectors, which will be introduced later.[14]*

The filled dots are data points of one class, the empty points data points of another. The maximum margin hyperplane is the plane that gives the greatest separation between classes. In this 2D case the hyperplane is a line, but we can imagine that for a 3D case the hyperplane would be a surface, and so on. In an N-dimensional case, the hyperplane is (N-1)-dimensional. The support vector are the points that lie closest to the hyperplane. So getting the maximum margin hyperplane involves making a plane that maximises the distance between the support vectors and that plane. The word 'distance' is key here.

What if the classes are not linearly separable, but can only be separated by a non-linear boundary. One way to solve this is by transforming the features. For example, we could transform the features (x_1, x_2) to $(x_1^3, x_1^2 x_2, x_1 x_2^2, x_2^3)$. This introduces many more features and makes the learning process much more computer intensive. This is where the strength of the SVM lies. Instead of transforming the features, it changes the definition of distance. Normally, this is of course defined by the inner product $\langle x, y \rangle$. By changing the definition of this inner product (or in SVMs terminology, the *kernel*), you curve space. In this way, finding the maximum margin hyperplane is still a linear problem, only in a non-linear, curved space. This does, however, mean that we can no longer easily access what the tool has done (i.e.

what the hyperplane is) because the function involved is now non-linear. This makes the tool a *black box*. There are many different ways we transform the inner product, many different ways we can choose a kernel. We will be using the most popular ones, where γ, r and d are tuning parameters:

1. linear: $\langle \boldsymbol{x_i}, \boldsymbol{x} \rangle$
2. polynomial kernel: $(\gamma \langle \boldsymbol{x_i}, \boldsymbol{x} \rangle + r)^d$
3. radial basis function (rbf): $exp(-\gamma ||\boldsymbol{x_i} - \boldsymbol{x}||^2)$
4. sigmoid: $\tanh(\gamma \langle \boldsymbol{x_i}, \boldsymbol{x} \rangle + r)$

So far we have assumed that the classes are separable. However, in real life applications, this is not often the case. If we apply our previous technique to non-separable data, we will definitely be overfitting, as no point is allowed to be classified incorrectly. This will lead to a very complex decision boundary. We need to introduce a penalty factor C. In the maximisation process, this factor (also called the complexity factor) punishes points that have been classified incorrectly. In the previous case, C $\to \infty$. Wrongly classified points were punished so severely that they cannot occur. In this thesis, we will treat C as a tuning parameter, optimising on the basis of test data.

3.7.6 Neural Networks

Neural networks are the final, and most powerful machine learning tool we will discuss. In the previous cases, we had our input attribute set that resulted in a specific output class via a certain function. Now more layers are added. We transform the features, to make 'new' features in our *hidden layer*, which are then transformed into new features in new hidden layers and so on. How does it work? We will first give a graphical explanation, after which we will shortly touch on the mathematics involved. The mathematical part of this section is based on ref. [12].

Figure 13 shows the basic architecture of a neural network with one hidden layer. We start with an input layer. The inputs here are the features, so if there are D features, there are also D inputs. Then, these features are combined and transformed to create new features, in the hidden layer. The amount of features in the hidden layer, or *hidden units*, does not have to be the same as the amount of input features. In our example, there is only one hidden layer, but in theory this process can be repeated many times, creating ever new hidden layers. The final step is the output layer. For binary classification, which we will be using, there are two options. Both these options will be used in this research. The first is to have two outputs, one for each class. Using a softmax function (the exact form of this function will be presented later), the values of these outputs can be transformed into a probability that a certain data point belongs to a class. For example, if output 1 is 0.7 and output 2 is 0.3, there is a 70% chance that the point belongs to class 1 and a 30% chance that it belongs to class 2. The second option involves having only one output. An optimal threshold value is manually placed on this output. If the output value is above that threshold you classify the point as one class, and if it is below the threshold you classify it as the other.

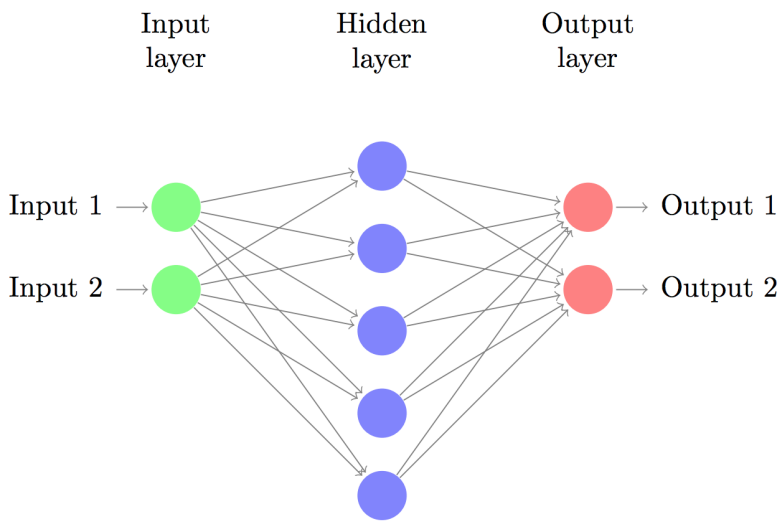


Figure 13: *visualisation of a neural network.*

Now we have a basic feeling of what a neural network is, we can look into it a little deeper by seeing how exactly the new units are created. As we saw in figure 13, we first transform D features to M hidden units in our first hidden layer. This is done by using activations a_j, defined as:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{3}$$

In this equation, $w_{ji}^{(1)}$ is called a weight. It determines how important a specific feature is in the new activation. The quantity $w_{j0}^{(1)}$ is a bias which sets off the activation. The activations can already be used as the hidden units, but they can also be transformed using an activation function h. After the activations have been transformed, we have M hidden units $z_j = h(a_j)$. These activation functions are mostly non-linear, which makes the neural network a black box algorithm, just like the SVM. This is because we would need to invert the whole network to get information like feature importance. Because of the complex equations involved, this is very challenging for small networks and nearly impossible for large ones. A few examples of often used activation functions are:

- sigmoid: $h(x) = 1/(1+\exp(-x))$ (note that this is a different sigmoid than the one used in SVMs)
- $h(x) = \tanh(x)$
- relu: $h(x) = max(0, x)$

These units function as the input for the next layer. Instead of x_i in equation (3) we take z_i and instead of summing up to D we sum up to M. This process can then be repeated until the desired amount of hidden layers is achieved. Then, the same process is done one last time, with the z_j's now being the outputs. If we have two outputs, we can use the softmax function $(\frac{e^{a_j}}{\sum_k e^{a_k}})$ as our last activation function. If we only have one output, any activation can be used, but mostly a sigmoid is chosen.

The architecture we have just discussed is the most basic form of a neural network. There are many other layers that can be added or alterations that can be made. We will only use one of

these in this research, and it has to do with overfitting. The technique we will be using is called *dropout*. As with random forests, we want to counteract overfitting by averaging the output over many different tools. The problem lies in the word different. The data going through the network or the architecture of the network could be different. However, the amount of data is often limited, so it is better to change the network architecture. This can also be hard, because finding optimal hyperparameters for each network can be difficult. We therefore use the same solution as we used with random forests. We remove a random subset of hidden units for each network. This makes each network different, greatly reducing the problem of overfitting.

The final concept we need to introduce is that of *epochs*. We train a neural network by sending the training data through it many times, each time tweaking the weights a little. This optimisation is done by the tool itself, using different types of optimisation algorithms, such as gradient descent algorithms. How many epochs you run depends on the problem. Generally we investigate how many epochs it takes for the performance to no longer improve. However, we should again watch out for overfitting. If you run many epochs, the tool might start learning the noise in the training data, which means that it will no longer perform well on other data. That is why you should also send validation data through the network after each epoch. This doesn't alter the weights, but tells you the actual performance of the network.

3.7.7 Performance measures

Now we have introduced the different algorithms, there is one final thing about machine learning we need to introduce. How exactly do we assess if a classifier is doing a good job? Well, there are many different measures that judge the performance of a tool. This is because there are two ways an algorithm can be wrong, and two ways it can be correct. This can be shown graphically in the following way.

Table 3: *The different ways a data point can be classified.*

		Truth	
		Class A	Class B
Prediction	Class A	True Positive (TP)	False Positive (FP)
	Class B	False Negative (FN)	True Negative (TN)

With these definitions we can define several ways to assess if an algorithm works well:

$$\begin{aligned} accuracy &= \frac{TP+TN}{TP+TN+FP+FN} \\ precision &= \frac{TP}{TP+FP} \\ sensitivity &= \frac{TP}{TP+FN} \\ True\ Positive\ Rate\ (\text{TPR}) &= \frac{TP}{TP+FN} \\ False\ Positive\ Rate\ (\text{FPR}) &= \frac{FP}{FP+TN} \\ Incorrect\ ratio &= 1 - accuracy \end{aligned}$$

These values can also be accompanied by an error. In what range can we expect the accuracy to be if we apply the classifier to a new data set? For accuracy (and therefore also for the incorrect ratio) this can be done with the following equation:

$$measure \pm 1.96 * \sqrt{\frac{measure(1-measure)}{n}} \tag{4}$$

Here 1.96 corresponds to a 95% confidence level that a new accuracy will lie in the given range (2σ). The amount of points in the test data is given by n. The derivation of this equation is less relevant and will not be discussed here. It can be found in chapter 5 of ref. [15]. It should be noted that this is an approximation on the actual error which has several conditions. First of all, the test set must be larger than 30. Second of all, each test point must be independent. Both these criteria will always be met in this research. The approximation works best if the accuracy (or incorrect ratio) does not come too close to 0 or 1. A rule of thumb is that the approximation works fine if $n * accuracy * (1 - accuracy) \geq 5$. This condition will also always be met, but it is still important to remember that for high accuracies, the approximation will work less well, and therefore the given error might not be exactly correct. For the other measures, calculating an error is less straightforward. We will assume that the error will have the same order of magnitude as the error for accuracy. As is now obvious, this approach relies on many assumptions and should therefore not be taken as absolute truth, but as an estimate.

In this research we will mostly be using accuracy, TPR and FPR. The other measures will be added for completeness. When the classes are balanced, meaning that there is roughly the same amount of data points in each of the classes, accuracy is a good measure of assessing

how well a classifier is doing. However, when the classes become imbalanced, this is no longer the case. Consider for example a problem with two classes, A and B, with 99 data points with class A and 1 data point in class B. A classifier that says: "Everything I see is class A", would get a 99% accuracy. Seems good, but that classifier is obviously useless.

Instead, a ROC-curve and the area under the curve (AUC) are used as measures. For our research, these measures will only be used in the case of the neural network with a single ouput, and we will therefore explain the concept in that setting. An example of such a curve can be seen in figure 14.

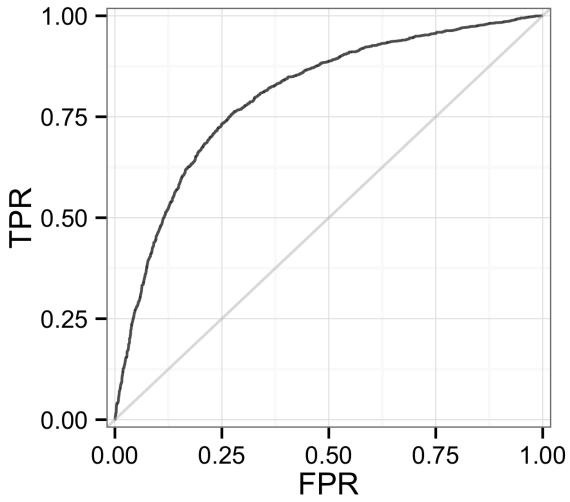


Figure 14: *Example of a ROC curve. The black line corresponds to a good classifier, the grey line to a random classifier.*

The ROC-curve relates the true positive rate (TPR) to the false positive rate (FPR). You might wonder how this could give a line in stead of only a point. This has to do with the threshold. Points with an output value lower than this were classified as a certain class (lets say negative) and points with an output value above it were classified as the other (lets say positive). A ROC-curve is created by varying the threshold from the lowest to the highest output, and each time calculating the TPR and FPR. For example, if you were to set the threshold at the lowest value, everything would be classified as positive, and so the TPR *and* FPR would be one. This corresponds to the top right corner of the curve. The same reasoning goes for the bottom right corner. If you set the threshold at the highest value, everything will be classified as negative, and the FPR and TPR would both be zero.

Obviously, a really good model would have a TPR of 1 and a FPR of 0. In figure 14, this is the top-left corner. Good classifiers result in ROC-curves which come close to that point. The AUC-measure is the integrated area under the ROC curve. If the classifier is good, and the curve approaches the top-left corner, this value would be high. If the classifier is bad, the area will be smaller and so the AUC will be lower. Does this measure have the same downfall as the accuracy with imbalanced classes? No! Because now, if it classifies everything as A, the amount of true negatives would be zero, and thus the FPR would be one, and the AUC would have a very low value.

3.7.8 Software

We have one final, practical thing to discuss. All these machine learning tools consist of complicated software, and we will definitely not be setting them up from scratch. There are several python machine learning packages of which we will use three.

- **Scikit Learn [11]:** This package will be used to train decision trees, random forests and neural networks.
- **TFlearn [16]:** This library is built on top of TensorFlow [17], another machine learning package, to make it more user friendly. This module will be used to train neural networks.
- **Keras [18]:** This library has a similar purpose as Tflearn, and will also be used to train neural networks. The differences between Keras and TFlearn are small. Both were used because of preferences by different collaborations.

4 Final state lepton classification

Now we have introduced all the concepts we will need, we can start talking about the actual research. As was stated before, the first part of the project will investigate if machine learning tools can be used to classify the leptons in the final state of our process (fig. 7) as being part of the OSSF pair or not. We will first look at our data. How was it created, and what features should we use. We will then see if machine learning is necessary, since maybe the two classes can easily be separated by hand. After we have understood the data properly and convinced ourselves that applying machine learning would indeed be worthwhile, we can continue onto training different tools. We will train several algorithms, tune them and compare the results. The tools we will be testing are: decision trees, random forests, support vector machines and single layer neural networks.

4.1 Data

One of the most important things when using machine learning is knowing everything there is to know about the data. How was it created and what do we know about the data points? What parameters can we use as features? Then we need to investigate how we can best pre-process our data, so it has a form we can use in machine learning. It should be noted that the data discussed in this section will be used as a basis for the training data for all tools. However, some modifications might be applied for a specific tool. This will then be discussed in the corresponding section.

4.1.1 Data simulation

We use simulated data, so no actual data from ATLAS or other LHC experiments. The same data was used as done in ref. [9], to make sure that results could be compared. We will therefore not go into the details of how it was created, but shortly touch on the most important steps. Several programs were used to simulate the LHC and the ATLAS experiment. The main program used is MadGraph5 [19]. It simulates the actual events (collisions). Then, Pythia [20] is used. What this program does exactly goes beyond the scope of this research. We can look at it as making the event more 'realistic', adding things like the hadron showers we discussed earlier, where quarks created in a certain process cause the creation of many mesons and new quarks. You might have noticed that I use the term *hadron shower* here instead of jet. This is because a jet is a single entity, the hadrons in the showers need to be *clustered* to form a jet. This is handled by FastJet [21][22].

How does MadGraph5 know how to simulate a process with SUSY particles? Such a process obviously depends on what SUSY model you are using. What are the masses of the particles, how do they interact? All this information can be fed to MadGraph5, which can then use it to simulate the desired process. One thing to note is that when producing a search strategy, it is important to create one that can be used on a wide range of models (MSSM with different mass ranges). When implementing the strategy in an experiment, you do not know what you are looking for, as any one of the many MSSM models could be the one actually found in nature. If you create a tool that is only sensitive to a small selection of models, chances are that it will not be sensitive to the actual model realised in nature. We will therefore train our tools on data originating from six different models. We will combine these, without telling the tool which is which. In that way, we hope that the tool will be applicable to all of them. The models we will take will have a Bino-like LSP and a Wino-like NLSP. They are distinguishable by their mass gap Δm. The mass gaps will be 15, 20, 25, 30, 35 and 40 GeV. If we assume

that our tool will only be usable for mass gaps in this window it will not be sensitive to models with larger mass gaps. This is not a problem, as this research is aimed at compressed spectra only.

4.1.2 Features

Each event has one opposite sign (OS) lepton and two same sign (SS) leptons. So, one lepton with a certain charge and two leptons with the opposite charge. One of the SS leptons belongs to the OSSF pair that originates from the Z-boson decay. This is the same sign pair lepton. The other comes from the W-boson decay and is called the same sign non-pair lepton. We will be classifying between pair and non-pair. The pair is the OSSF pair constructed by the OS lepton and the SS pair lepton. The non-pair is the OSSF pair constructed by the OS lepton and the SS non-pair lepton.

The data from the simulation is not in the correct form to be used in machine learning. The MadGraph output contains a lot of kinematic information about all particles in the process. We need to choose what information will be useful to us, i.e. what our features will be. We want to classify charged leptons, so we only need the kinematic information from these. The kinematic information of a particle is contained in its 4-vector: $(E, \vec{p})$. Each lepton only has three degrees of freedom ($\vec{p}$), because the mass of the leptons is very small compared to the momentum. So, $E = \sqrt{m^2 + |\vec{p}|^2} \overset{\text{m}\ll\text{p}}{\approx} |\vec{p}|$, which means that E is not independent. So, with three features, a lepton is completely defined. However, we will need more features than that to get the best classification, because combinations and transformations of these three features could create simpler decision boundaries, leading to better classification. A very advanced machine learning tool would be able to construct all these other variables by itself, but the tools we will be using need a little more help. The combinations and transformations all lead to different kinematic variables.

The first set of variables we will choose are the ones used in ref. [9] to classify the OSSF pair: ΔR and M_{l+l-} (for definitions of these variables see sec. 3.5.1). The third variable used in that research was $\max(\vec{p}_T)$ (selecting the lepton pair with the highest summed $\vec{p}_T$). We will change this to $\Delta|P_T|$ (the difference in size of the two candidate pair leptons). These variables have already proven to allow for decent separation of the two classes. It is therefore a logical step to start with them, and see if machine learning tools can improve the classification accuracy by combining and transforming them. These three variables are not the only ones we will use. The next two variables we will add are $\Delta\phi$ and $\Delta\eta$. They are already contained in ΔR, but we want the tool to have access to them separately as well. The final features we will add are the 4-vectors of the individual leptons. We will add the 4-vectors of the OS lepton and SS non-pair lepton to the non-pair attribute set and those of the OS lepton and SS pair lepton to the pair attribute set. You will notice that we have added the OS lepton 4-vector to both attribute sets. It is only there to provide a frame of reference.

We have now chosen what our features will be, and can extract the information we want from the simulated data. For each event the two OSSF candidates will be identified, and their kinematic information will be extracted. The pair data point will get label 1 and the non-pair data point will get label 0. Knowing which of the candidates is the pair is important because we want to perform supervised machine learning. This information can be extracted because MadGraph5 also gives the particle from which the final state is created. We have now completely processed our data to be ready for implementation. The shape of our data can be seen in table 4.

Table 4: *The shape of the data set. Each attribute set consists of 13 features. In the training data, the label is provided. In the test data, the attribute set is provided to the tool which then gives it a label.*

#Datapoint	**Attribute Set**							**Label**
1	$\Delta\phi_1$	$\Delta\eta_1$	ΔR_1	$M_{l+l-,1}$	$\Delta P_{T,1}$	$P^{\mu}_{SS,1}$	$P^{\mu}_{OS,1}$	1
2	$\Delta\phi_2$	$\Delta\eta_2$	ΔR_2	$M_{l+l-,2}$	$\Delta P_{T,2}$	$P^{\mu}_{SS,2}$	$P^{\mu}_{OS,2}$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	$\Delta\phi_n$	$\Delta\eta_n$	ΔR_n	$M_{l+l-,n}$	$\Delta P_{T,n}$	$P^{\mu}_{SS,n}$	$P^{\mu}_{OS,n}$	1

As we mentioned before, it is important to split our data in training, validation and test sets. In this first classification problem, we will only use two sets. We will use the same sets for validation and testing, and call it the test set. This is often done when the danger of overfitting on the validation set through hyperparameter tuning is small. We believe this to be the case here, as the amount of hyperparameters available in the implemented tools is relatively small.

4.1.3 Graphical representation

We are now ready to train some algorithms! But before we do this, we must ask ourselves if this is worth doing. Can it not easily be done by hand? We will do this by plotting the normalised distributions of our first three features. This will serve to convince ourselves that classification by hand is indeed problematic and simultaneously to confirm the results presented in ref. [23].

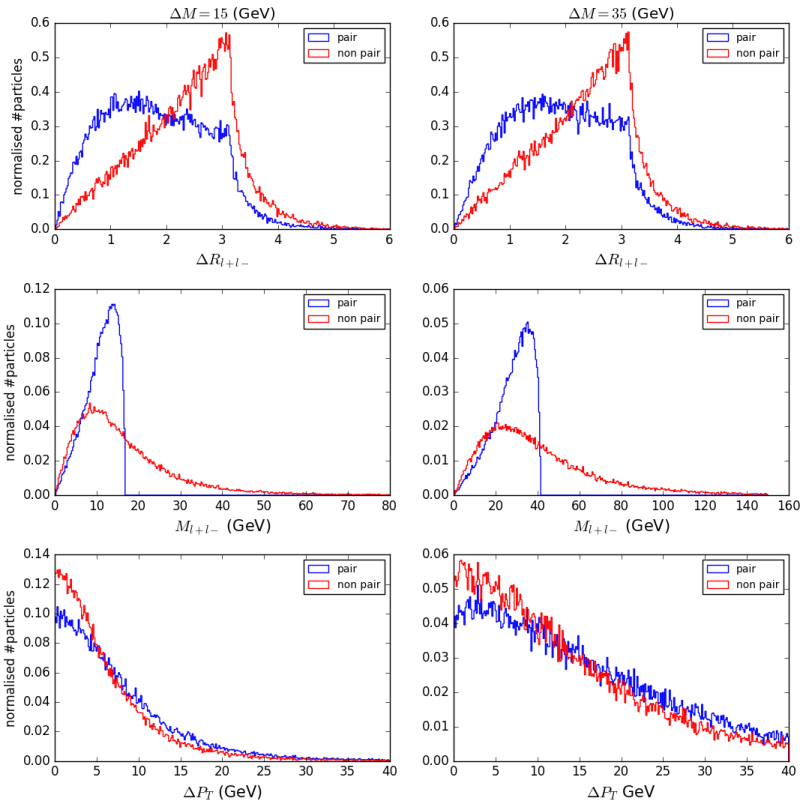


Figure 15: *Visualisation of three features (ΔR, M_{l+l-} and ΔP_T). The column on the left shows the normalised distribution for data produced from a model with $\Delta M = 15$ GeV. The column on the right corresponds to data from $\Delta M = 35$ GeV.*

We see that the normalised distributions of the first two features confirm the results in ref. [23].

The figures showing the distribution of the ΔR_{l+l-} indicate that the pair-lepton-distance is inclined towards lower values, in contrast to the non-pair-lepton-distance, which is often larger. Classifying on the lowest ΔR_{l+l-} is therefore a good idea. The same holds for M_{l+l-}. The pair tends to have a lower invariant mass than the non-pair. The ΔP_T plot also confirms that classification by hand is not easy. The distributions of the pair and non-pair are very similar, with a lot of overlap. We can see that the non-pair leptons tend to have P_Ts that lie closer together. All classification methods are fallible. As we can see in the plots, the amount of overlap is quite large, especially for the transverse momentum. This means that not all classifications will be done correctly, exactly what we expected. We also confirm the findings that the accuracy decreases with higher mass gap. The plots show that the overlap increases when $\Delta M = 15$ GeV $\rightarrow \Delta M = 35$ GeV.

4.2 Training

4.2.1 Decision Trees

We will start with the most basic tool, the decision tree. We will first investigate the performance of the tool with an attribute set that does not include the raw 4-vectors. Adding fewer features makes for a simpler tool, and we first want to see if the other features are necessary. We will investigate what the optimal tree depth is. Then we will add the 4-vectors, to see if this indeed improves the performance. Finally we will add manually transformed features. In all these steps we will also investigate what the features' importance is.

First we want to determine what depth the tree should have for the best result. This will be done by training the algorithm for different depths and then plotting the accuracy with respect to the tree depth. Each depth will be analysed with ten thousand training points and ten thousand test points. The error bars were created using equation 4. The result of this can be found in figure 16. We find that the optimal maximum depth is around nine. Thus this will be the value we will use. From a max. depth of one to eight, the tool can improve the accuracy by adding new cuts and thus making the decision boundary more complex. When the tool is allowed to become deeper than about ten cuts, overfitting kicks in. The tool starts to make a too complex decision boundary, that only works for the training data. On the test data, it decreases the accuracy.

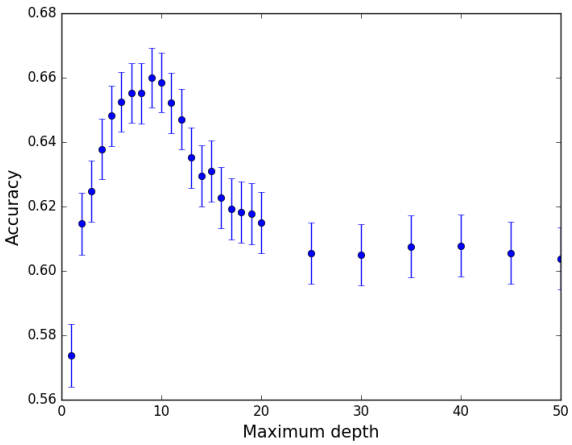


Figure 16: *This plot shows how the accuracy depends on the max. depth. Each max. depth was trained with 10,000 training points and tested with another 10,000 points. The optimal value lies around nine.*

Now we will train our tree with max. depth op nine with sixty thousand training points and test it with another sixty thousand. The results of this can be found in table 5.

Table 5: *Result of a decision tree with a max. depth of nine trained with 60,000 training points and tested with 60,000 test points*

Accuracy	**Sensitivity**	**Precision**	**TPR**	**FPR**	**Incorrect Ratio**
0.671 ± 0.004	0.708	0.657	0.708	0.366	0.329 ± 0.004

Now we can ask ourselves what features are most important. Plotting the feature importance in a bar diagram gives the result found in fig.17(a).

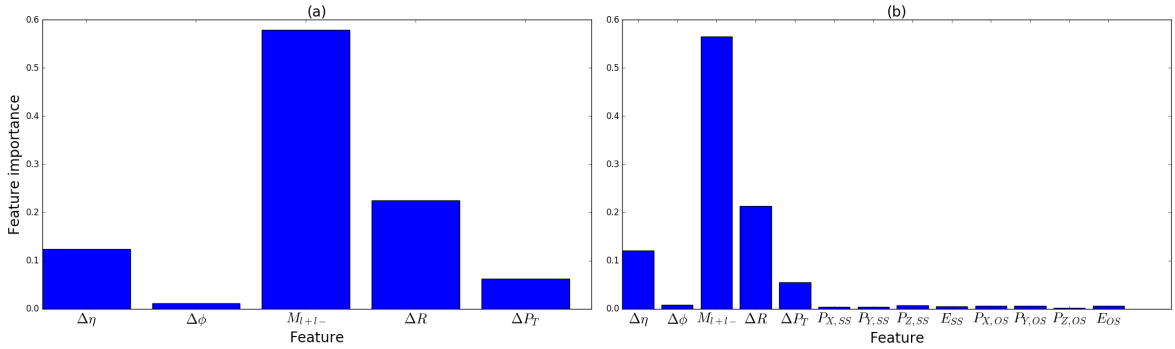


Figure 17: *(a) The x-axis shows the five features that are used for training. The y-axis shows the importance of a feature as a fraction of the total. We can clearly see that the invariant mass is the most important feature, followed by the ΔR and $\Delta\eta$. (b) The x-axis shows that we have increased the amount of features to 13.*

As we can see in the plot, the invariant mass is the most important feature, followed by the ΔR. That the mass is a good feature was to be expected. We know from our data visualisation plots that there is a very sharp cutoff at a certain M_{l+l-} for the pair points, but not for the non-pair points. By classifying everything above this cutoff as non-pair, many points will already be classified correctly. If we expect the algorithm to have classified using this statement, we should realize that our tool will become less useful at higher mass gaps, as the amount of points above the cutoff will then be smaller. This is not a problem, since this research is aimed at classifying data from compressed SUSY models. From our visualisation plots we can also conclude that ΔR should indeed be a good parameter to classify on. That $\Delta\eta$ is also a good feature proves that we cannot expect a decision tree to make useful features itself. The $\Delta\eta$ feature doesn't add any new physical information, but is still a useful feature.

What can we do to improve these results? First, we can try and add more features. We start by now using the complete attribute set presented in table 4. Learning the algorithm with this new attribute set changes the accuracy slightly from 0.671 $\rightarrow$ 0.669 (at a max. depth of 9). This lies within the margin of error, so we can say that for decision trees, adding new features does not improve the performance of the classifier. Figure 17(b) makes this plausible, because we see that the decision tree is just not using any of the new features.

A problem with decision trees is that the algorithm is linear. It takes a certain cut in a certain feature. But what if that cut should be quadratic? The algorithm cannot search for these type of relations. However, we can help it by adding functions of the features ourselves. First, we will try to include linear combinations of the features to our attribute set. With the

added eight features from the four-vectors our attribute set now has a size of 13. We have seen that the new features are not useful on their own, but they might be in combination with others. Taking all linear combinations with two features (e.g. $\Delta R + M_{l+l-}$) will give $13+12+11+..+2+1+13 = 104$ features,including the original 13. This means we count $2\Delta R$ as different from ΔR. This might not be necessary, but it can't hurt either. Next we will try powers (e.g. ΔR^n). We will use $n = \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, 2, 3, 4$. This will give us $13 * 6 + 13 = 91$ features. Finally, we will combine these approaches. This means we will also take the nth power of the linear combinations. This will give us $(104) * (6 + 1) = 728$ features. For all of these attribute sets we will use a max. depth of eight (we have seen that the difference between a max. depth of nine and eight is minimal). We do this, because we are looking for a new feature that will improve our accuracy. If such a feature exists in our new attribute sets, the depth does not need to be larger to find it. A good feature will be picked out by the algorithm in the first few layers. This gives us the results presented in table 6.

Table 6: *Results of a decision tree trained with different sets of features added.*

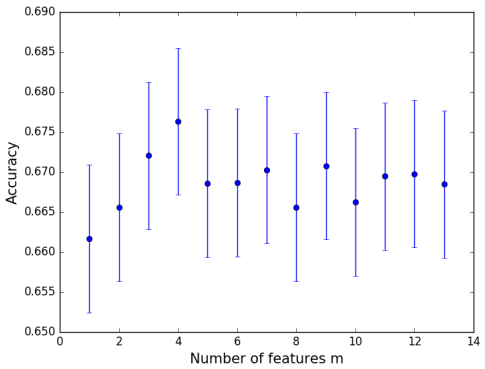
	Lin. Combinations	**Powers**	**Powers of Lin. Comb.**
Amount of features	104	91	728
Accuracy	0.67 ± 0.01	0.66 ± 0.01	0.67 ± 0.01
Most important feature	$2M_{l+l-}$	M^4_{l+l-}	$M^{1/3}_{l+l-}$

As we can clearly see, this does not change the results. All differences are within the margin of error. The accuracies are all fairly similar, comparable to the results obtained without manually adjusting the attribute sets. The most important feature is also still (a function of) the invariant mass. Apparently, there are no new, better features in our new attribute set. The trained algorithm will therefore probably also be similar. The fact that the most important feature is still a function of the invariant mass strengthens this claim. If an algorithm works with M_{l+l-} it will obviously also work for M^n_{l+l-}.

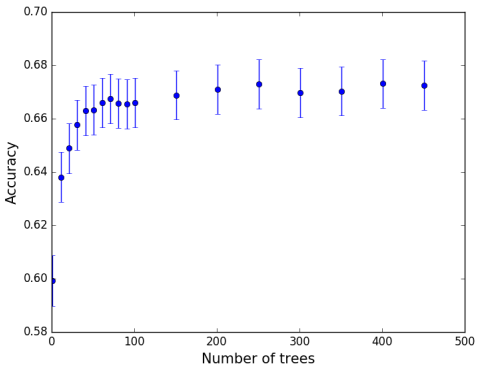
4.2.2 Random Forests

In this section we will use a random forest algorithm to classify our data. We will take the number of trees and the amount of features used per tree (m) as tuning parameters. We will use the attribute set with thirteen features described as in the previous section. One might argue that we should also look at the functions of the features (linear combinations etc.). However, looking at them did not improve the decision tree verdict, and will therefore not improve the majority vote of those trees. If each separate tree is worse in deciding, the total decision will also be worse.

The results of training forests with 10,000 training points and 10,000 test points are as follows:



(a) *This plot shows how the accuracy depends on the number of features used per tree in the forest. There seems to be no significant dependence. Here 100 trees were used.*



(b) *This plot shows how the accuracy depends on the number of trees used per tree in the forest. We can clearly see that the performance improves up to about a hundred trees, when it starts to level off. There might be some slight improvement after this, but this is not significant. The point on the left is the single-tree-case. The number of features was set to four.*

Figure 18: Dependence on tuning parameters

First of all, we can gather from figure 18a that the accuracy does not significantly depend on m. There seems to be some improvement from one to four features, after which the accuracy decreases and then levels out. This would make sense, because when too few features are used, the individual trees do not have enough information to give a good classification. When too many features are used, the trees are not uncorrelated and so overfitting sets in. Second of all we look at figure 18b. Here we can see that a random forest is an improvement over a (non-tuned!) tree. The accuracy decreases when the number of trees goes to one. When the number of trees is around a hundred, the accuracy is more or less stable. This means that adding more trees does not improve the algorithm. If we include these findings in another algorithm, that is trained with 60,000 training points and tested on 60,000 test points, we obtain the results presented in table 7. We have used $m = 4$ and $\#trees = 100$.

Table 7: *Result of decision tree with 60,000 training points and 60,000 test points*

Accuracy	**Sensitivity**	**Precision**	**TPR**	**FPR**	**Incorrect Ratio**
0.685 ± 0.04	0.707	0.676	0.707	0.337	0.315 ± 0.04

This is not significantly better than the decision tree. How can this be, didn't we see in figure 18b that there should be an improvement? Yes, but that was with respect to a non-tuned

tree. The main advantage of using a random forest is that you do not have to worry about overfitting. However, we already implemented some techniques to counteract this phenomenon when we were looking at decision trees, namely tune the max. depth of a tree. Apparently, this was done to such an extent that overfitting was no longer a real issue, and therefore the random forest algorithm was obsolete.

4.2.3 Support Vector Machines

We have seen that our linear classifiers give quite poor performances. We will therefore now turn to a non-linear classifier, the SVM. We will first see which kernel we should use and which values we should give its hyperparameters. We have four options for the kernel: linear, polynomial, rbf and sigmoid. We will calibrate their corresponding hyperparameters by running the algorithm on a range of reasonable hyperparameter values, and finding the best value by making a colour plot where the hyperparameters are the variables. The colour of the plot indicates the accuracy. The accuracy was calculated by testing the algorithm on independent data. The linear kernel will be treated differently. We will not need to find optimal hyperparameter values because the linear kernel doesn't have any.

The results are shown in figures 19 and 20. Because the rbf kernel is only dependent on one parameter, we will find the optimal value for the complexity parameter (also known as the *penalty*) using this kernel. It will be treated as just another hyperparameter. We only need to find the optimal value for this parameter once because the complexity of the system is of course independent of which kernel we choose to analyse it. We also notice that the degree d in the polynomial kernel was not used for optimisation. This can be justified by the fact that the accuracy will improve if the degree gets larger. This is because when the degree increases so does the dimensionality of the feature space. The feature spaces of kernels with lower degree are part of the feature space of kernels with higher degree. So, a higher degree can only improve the accuracy. However, a higher degree also means a higher computation time. We found that d=3 gave the highest still acceptable computation time, and so this was the fixed value we chose for the polynomial fit.

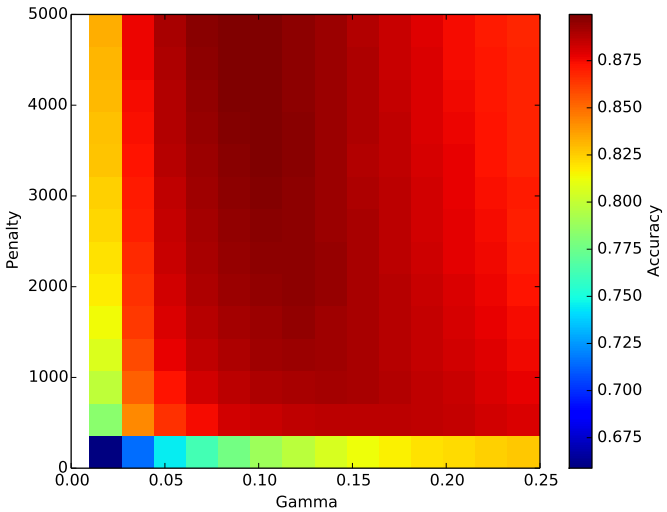
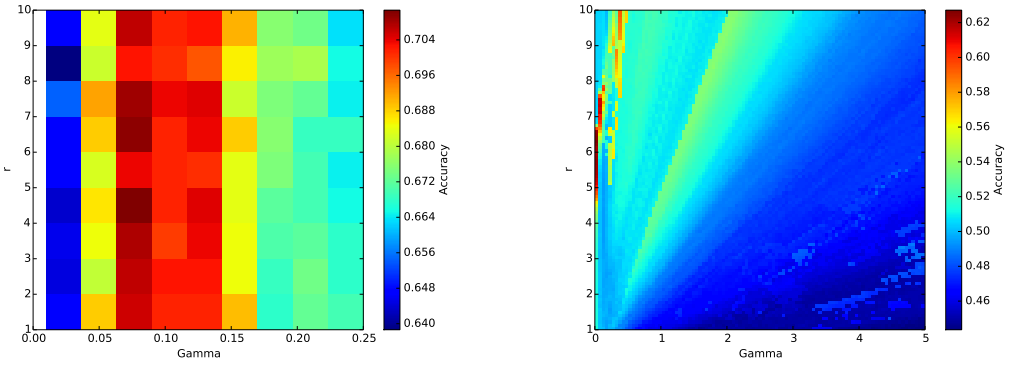


Figure 19: *Accuracy as a function of the parameter γ and the penalty (complexity) parameter C using the rbf kernel:* $\exp(-\gamma||x_i - x||^2)$. *The algorithm was trained with 40,000 training data point and tested with an independent batch of 50,000 test data points.*



(a) *The dependence of the accuracy on r and γ, using a polynomial kernel of order three with $C = 500$: $(\gamma\langle \boldsymbol{x_i}, \boldsymbol{x}\rangle + r)^3$.* The algorithm was trained with 5,000 training points and 5,000 independent test points.

(b) *The dependence of the accuracy on r and γ, using the sigmoidal kernel, with $C = 500$: $\tanh(\gamma\langle \boldsymbol{x_i}, \boldsymbol{x}\rangle + r)$. The algorithm was trained with 10,000 data points and tested with an independent set of 10,000 test points*

Figure 20: *The dependence of the accuracy on different tuning parameters for different kernels.*

We immediately see that the rbf works best, achieving an accuracy of almost 90%. The polynomial kernel produces a maximum accuracy of ∼70%. The sigmoidal kernel works worst of all, with a maximum accuracy of ∼60%. The same goes for the linear kernel (this was of course to be expected given the results of linear classifiers so far). It should be noted that the amount of training points varies per kernel. This was done to keep computation time reasonable. However, it might be that the polynomial kernel performs worse than the rbf kernel because it had less training examples. Future research should check this. From the rbf we can conclude that the tools work best for large values of C. This means that the decision boundary can be very intricate. When this is the case, we should always worry about overfitting. However, the accuracy was calculated with an independent set of data, so it is implausible that overfitting has occurred. We also see that after about C=500, the accuracy increases only slightly, with about two percent points. Increasing C, and therefore the complexity, does increase computation times though. We will therefore choose to set C=500, because doing so keeps the computation time at an acceptable level, whilst still giving a very accurate result. We will however keep in mind that increasing the complexity will most probably give an even better result. The kernel parameters are:

- kernel=rbf
- C= 500
- $\gamma = 0.1$

Training this algorithm with 100,000 points and testing it with a separate 100,000 points gives the result shown in table 8. We see that the accuracy is very high, much higher than in the non-linear case. Apparently, the decision boundary is non-linear in our feature space.

Table 8: *Results of a SVM with hyperparameters set to the values mentioned above, trained on 100,000 points and tested on another 100,000 points.*

Accuracy	**Sensitivity**	**Precision**	**TPR**	**FPR**	**Incorrect ratio**
0.916 ± 0.02	0.972	0.874	0.972	0.140	0.084 ± 0.02

Now one might interject and ask why we include 13 features in our attribute set? It is true that in the section on decision trees we concluded that only a few features were relevant. Why do we now include all those features that seemed to be useless? Well, a nonlinear classification algorithm might use very different features. We can test this by training the

same SVM algorithm as before, only now using only the features that were important in the decision tree algorithm. These were in decreasing order of importance: $M_{l+l-}, \Delta R, \Delta\eta, \Delta P_T$. The accuracy of this algorithm is only 66%, way below the value found when using the entire attribute set. This is interesting, because it means that the additional features are important, but a non-linear algorithm is needed to extract the useful information!

4.2.4 Single Layer Neural Network

The final approach we will be looking into is the neural network. In this first project, we will only consider neural networks with one hidden layer. This leaves a few hyperparameters we can choose when constructing our neural network. These include the amount of hidden units in the hidden layer (M) and the activation function that is used. Furthermore we need to set the number of epochs and the size of the training, validation and test data sets. The bigger the better applies to all of these last parameters, and we will therefore not go into them in detail, only mentioning what value they were set to.

In table 9, the results of neural networks trained with different activation functions are displayed. We see that the sigmoidal activation function works best, with an accuracy of 91.2%. The tanh function is not far behind with an accuracy of 86.1%. The linear function does not work well, only giving an accuracy of 61.7%. This was to be expected, as all previous results seem to indicate that our data is not linearly separable.

Table 9: *Neural network with one hidden layer with 1000 units. It is trained with 50,000 data points and validated with a separate set of 50,000 data points. The algorithm was assessed using a separate test set of again 50,000 data points.*

Function	**Accuracy**	**Sensitivity**	**Precision**	**TPR**	**FPR**	**Incorrect ratio**
Linear	0.617 ± 0.04	0.741	0.595	0.741	0.507	0.383 ± 0.04
Sigmoidal	0.912 ± 0.03	0.920	0.905	0.920	0.097	0.088 ± 0.03
Tanh	0.861 ± 0.03	0.911	0.830	0.911	0.189	0.138 ± 0.03

4.3 Conclusions

We have applied several machine learning tools to our training data. All models show some potential for classification, though some do a better job than others. The most important question is if they do a better job than the classification by hand. We will compare results of three different techniques developed in ref. [23]. We will also compare it to the technique used by ATLAS, where the lepton pair that has an invariant mass closest to the Z-mass is chosen. We will test our machine learning tools by selecting the best performing tools from each algorithm, and testing them with data produced by a single mass gap. So far we have tested the tools with testdata that had the same shape as the training data, so with a mixture of mass gaps. However, because in nature there is only one 'true' model, we now need to use only one mass gap to get a physically relevant result. The results of this can be seen in table 10.

Table 10: *Comparison between classification accuracies produced by hand (first 4 rows) and by machine learning tools (last 4 rows) for different mass gaps.*

Algorithm	Δm =**20 GeV**	Δm =**30 GeV**	Δm =**40 GeV**
$\min_{i,j} \lvert M_{l_i,l_j} - M_Z \rvert$	0.38	0.39	0.43
$\min_{i,j} \Delta R(l_i l_j)$	0.66	0.64	0.62
$\max_{i,j}(P_T(l_i), P_T(l_j))$	0.54	0.52	0.52
$\min_{i,j} M_{l_i,l_j}$	0.62	0.62	0.60
Decision Tree	0.664 ± 0.009	0.661 ± 0.009	0.687 ± 0.009
Random Forest	0.693 ± 0.009	0.676 ± 0.009	0.706 ± 0.009
Support Vector Machine	0.916 ± 0.005	0.909 ± 0.005	0.904 ± 0.005
Neural Network	0.920 ± 0.006	0.907 ± 0.006	0.873 ± 0.007

The results are mostly as we would expect. At low mass gap, the decision tree and random forest, both linear classifiers, do comparably well to the best result from classification by hand (min ΔR). This makes sense, because a decision tree does the same as a human does when classifying, drawing straight lines through feature space. We may thus say that with these features, an accuracy of around 65% is the best that can be done by hand. The decision tree and random forest do exhibit some odd behaviour. Their accuracies increase for higher mass gaps, having a dip at $\Delta m = 30$ GeV (especially for the random forest). We expected the opposite (see sec. 4.2.1). However, the effect is quite small, almost within the margin of error.

For a better result, we need new features, or non linear cuts in this feature space (which comes down to the same thing). This is what SVMs and NNs do. We can clearly see that these improve the results greatly, increasing the accuracy to more than 90%. We have shown that ML can indeed be used to classify final state leptons as pair and non-pair, provided that the tool used is non-linear. We can qualitatively say that using ML tools in this application could definitely be beneficial, and advise that these tools should be incorporated in future search strategies.

5 Signal-Background Classification

The second project we will be embarking upon is aimed at classifying simulated LHC events as signal or background using deep learning (using a neural network with more than one hidden layer). We will follow the same basic structure as in the previous section. First, we will look at how our data was generated and what the features should be. We will then ask ourselves if it is indeed necessary to resort to machine learning. Then, we talk about pre-processing our data, which needs a bit more attention than in the previous project. We will talk about the architecture of our neural network, and finally evaluate the performance.

5.1 Data

5.1.1 Data simulation

We will again use the same data described in ref. [9]. With the OSSF-pair classification, we used all the information that the simulation provided us with (see section 4.1.1). However, this is not very realistic. At the LHC, you can't directly 'look' at the event, you need to measure the outgoing particles with a detector. As we have seen, the ATLAS detector is not infallible. For example, it cannot measure particles that only interact weakly, and it has all types of efficiencies for measuring particles. To create a classifier that can be used in the real world, it needs to be trained on this type of data. This can be achieved by adding another step to the creation of the data. After the event has been simulated, a detector simulation is added to produce the data we need. The detector simulation used is Delphes [24], implemented in MadAnalysis5 [25].

Another difference with the previous project is that we are now classifying events, instead of particles in a specific event. We want to separate background from signal. This means we now also need to consider background processes. Background processes are defined to have the same final state, $3l + \not{E}_T$. Processes that have exactly this final state from the hard scattering event are called irreducible backgrounds. Processes that have a different final state from the hard scattering event, but appear to have the same final state in the detector are called reducible. These processes appear to have the same final state due to mismeasurements. Our ultimate goal is to compare our classification performance with that of ref. [9], which employs a strategy based on human produced cuts. We would therefore like to use the same backgrounds as were used in that project. However, there is a problem for reducible backgrounds. The mismeasurements that cause the background to have the same final state don't occur very often. This means that only one in many reducible background events actually looks like the signal. However, these processes occur so often in the LHC, that even a tiny fraction of these events leads to a substantial amount of events that look like the signal. This is where the problem lies. Simulating events is computationally intensive, and takes time. If we want many events which look like the signal (a large data set is something that is required in machine learning), we will need to produce an enormous amount of events for each reducible background. Because of practical and time-related constraints, this was not an option. We have therefore chosen to only use the following backgrounds: WZ, ZZ and WWW. The ZZ background is not technically irreducible, because it has a $4l$ final state. However, because the mismeasurement it requires to look like the signal involves ATLAS to not measure a lepton instead of creating a fake lepton, we have enough of these events to include them.

5.1.2 Features

Now we know where our data comes from, we can focus on what information we want from the simulation to use as features. In contrast to the previous project, we will now use a more advanced machine learning tool, that has proven to be able to create higher-level kinematic variables from low-level ones [26]. Thus, it is not necessary to add variables like the invariant mass, we can just use the raw 4-vectors. We saw earlier that the energy is not an independent variable, and therefore, we will not use it. Instead of using p_x, p_y, p_z we will use P_T, η, ϕ, which carries the exact same information. The first features we will choose are the kinematic variables of the three charged leptons. We will also add their identities ((anti)electron or (anti)muon). This results in 12 features. We now turn to the other part of the final state, the $\not{E}_T$. The missing transverse momentum is a vector, perpendicular to the beam-pipe. Its direction is therefore set by ϕ. Its magnitude can be transformed to give us the $\not{E}_T$. This ϕ and $\not{E}_T$ will both be added as features. The final variables we will add as features come from the jets. Jets are not part of the hard scattering final state, but can be created in processes that occur before the hard scattering event. The amount of jets can vary per event. Each jet has a kinematic 4-vector. We will add the number of jets and the P_T of the jet with the highest P_T as features. If an event has no jets, the P_T (jet) will be set to zero. Adding the other kinematic variables or the 4-vectors of additional jets would lead to too many features. This gives us a total of 16 features. The labels of the data points are now signal (1) or background (0).

5.1.3 Parameterised Neural Networks

Just like in the previous project, we will again be training with different SUSY models, to make the network more versatile. In this project, we will use 21 models with different $m_{\widetilde{\chi}_2^0}$ and $m_{\widetilde{\chi}_1^0}$, all with a Bino-like LSP and a Wino-like NLSP. Which masses we will use exactly will be discussed later. Previously, we just added all the data from the different models together and trained a classifier on that. This is however not the only way to do this. Another way is by parameterising. We add model parameters to the features before training. These model parameters are the masses of the lightest and second lightest neutralinos. This brings the total amount of features up to 18. The idea of a parameterised neural network is that the classifier will learn to adapt its search strategy depending on the model it is fed data from. It has been shown that parameterised neural networks can improve the classification performance compared to that of networks trained on data from a mixture of models without the added model parameters. Parameterised networks can also learn to interpolate between the models, so that it can also be used to classify data from models that weren't used in training [27].

Now it should be noted that in real data, we don't know what SUSY model we are looking for. We just get a bunch of measurements from ATLAS, and the classifier is used to separate the signal from the background. But you do not know which signal has to be separated. So you present the probability that you have excluded something **given** a certain model. In a parameterised neural network, this is done by using the measured data as the first 16 features, than adding the model parameters as features manually for each model separately, resulting in an exclusion significance.

5.1.4 Pre-processing

The data is now in the correct shape, so we could in principle perform machine learning on it. However, there are a few last adjustments that need to be performed. First of all, it is better to transform all the data such that their mean vanishes and their variance tends to 1 ($\mu = 0, \sigma = 1$). We will not discuss why this is exactly. It is enough to know that it improves the neural network algorithm. We will take the average and standard deviation of each feature based on all the data (so signal and background). We will then perform the following transformation on the features, where μ is the mean of the old features and σ the standard deviation:

$$\text{new feature} = \frac{\text{old feature} - \mu}{\sigma}$$

The next step involves splitting the total data set in three, producing a training, validation and test set. The training set will consist of events from the three background processes together with events from a mixture of all the signal models. This means that there will be only one training set. The validation set is exactly the same, only made up of different data points, to check if the model is not overfitting. There will, however, be more than one test set. We want to know how our classifier works on real data so we have to test on data that is realistic. In nature, only one SUSY model can be correct, so you need to test the classifier on each model separately.

The final step is adding weights. When producing the data, you ask MadGraph5 to produce a specific amount of events per process. The problem is that the ratio between these amounts doesn't have to be physically accurate. For the test set, this is a problem (we will talk about the training and validation sets later). We want these data sets to resemble real data as much as possible, and thus the ratio between the different events should be correct. How can we achieve this? MadAnalysis5 calculates the amount of events per process normalised on luminosity using the cross section. Luminosity is a measure of how many collisions occur in the LHC. Of course, this is equal for all processes, and so we need to normalize on this value, using the cross section to set weights. It should be noted that this does make our test data very imbalanced (see figure 21), which means we cannot use accuracy as a performance measure.

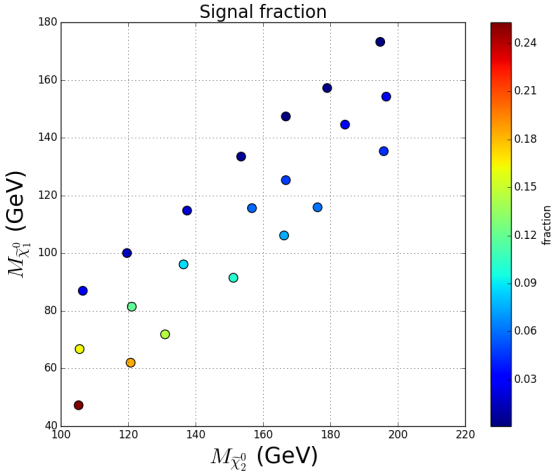


Figure 21: *Visualisation of the composition of the test data. We can see that the data set is indeed very imbalanced, with small fractions of signal data. For smaller mass gaps and higher particle masses, this fraction decreases.*

We will go a slightly different route for the training and validation data. It doesn't have to resemble real data as closely, it just needs to give the machine learning tool the correct information. In machine learning problems, it is mostly a good idea to give it the same amount of data points per class. So, we want the same amount of signal as background events. How about the ratios within the classes? We will take a different approach for the signal and background processes. For the background processes, we will use the physical ratios in the same way we weighted the test set. The ratios between these processes are known, so it is better to use them. If we were to, for example, have the same amount of events for the WWW and WZ processes, the overall performance would probably suffer, because the WWW background is much smaller than the WZ one. Within the signal class, this argument doesn't hold. The tool needs to be good at classifying all signal processes, because we do not know which process is the correct one. If we were to add realistic weights (so normalised on luminosity), there would be few examples of a model with a small cross section, which would make the tool less sensitive to it. This model might just be the one found in nature.

5.1.5 Graphical Representation

We will again convince ourselves that machine learning is necessary by representing the distributions of the features in 1D histograms. This is also useful because it will give us an intuition for the shape of the data. Displaying plots for each signal process would be a bit of an overkill, as we have 16 features and 21 models, which would give us more than 300 plots. We will only present one set of plots here, belonging to the model point with $m_{\tilde{\chi}_2^0} = 157$ GeV and $m_{\tilde{\chi}_1^0} = 116$ GeV. The background processes will be combined and weighted according to there physical ratios.

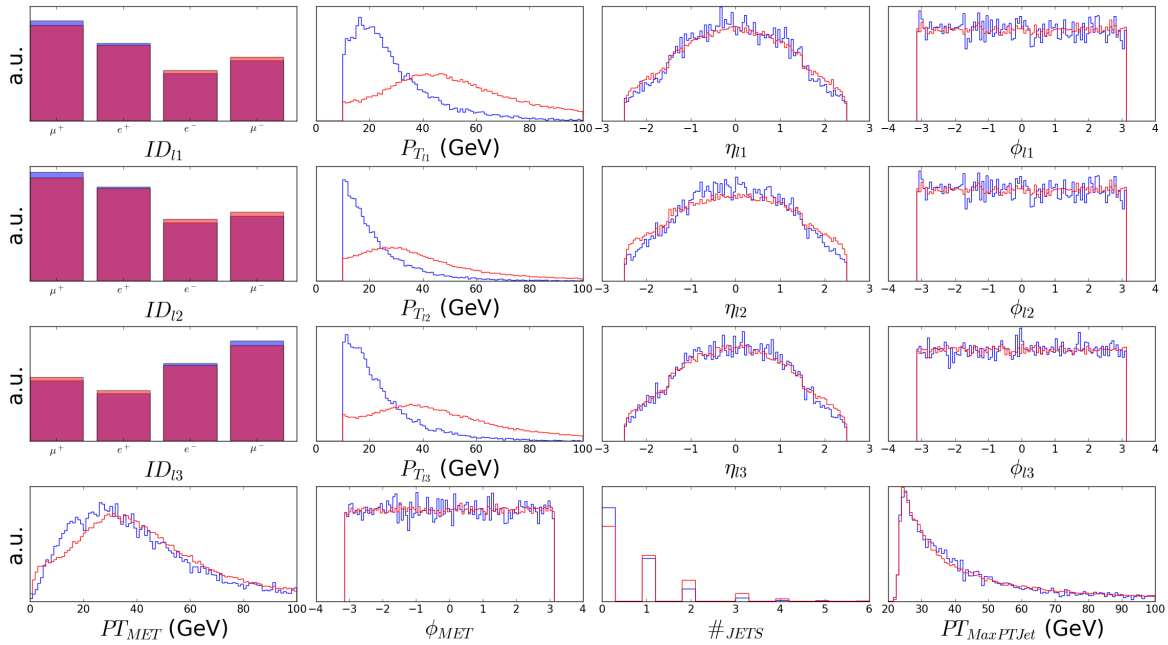


Figure 22: *Feature visualisation of the model point with* $m_{\tilde{\chi}_2^0} = 157$ *GeV and* $m_{\tilde{\chi}_1^0} = 116$ *GeV. There are 16 features in total. The red line represents background and the blue is signal.*

We can see that there is no clear way to separate the two classes. The transverse momenta of the three final state charged leptons do have some separation, but there is also still substantial overlap. There is also some separation in the amount of jets: background processes tend to have $\#jets > 1$ more often than signal events. Of course, it was to be expected that there would be a lot of overlap. Existing search strategies are all quite complex and don't lead to

total separation. However, it is important to confirm this for ourselves. One final thing to note in these plots is the cut that is visible for the lepton P_T at 10 GeV. Leptons with lower P_T have not been simulated (see ref. [9]).

5.2 Training of the Neural Network

We can now start training a network. In the previous project, training mostly involved tuning hyperparameters. In a deep neural network (DNN), there are many hyperparameters to tune. Each hidden layer can have a wide range of units and activation functions. The overall depth of the DNN can also be varied. We can add dropout layers. In this research, we will not be tuning any of them. We will take a fairly standard DNN, and train our data on that single one. This is because we want to show that even without tuning, there are impressive performances to be obtained. There is one thing that needs to be tuned, and that is the number of epochs.

5.2.1 Architecture

The architecture of the neural network we will be using is fairly standard. We start with our 16/18 input features (for the non-parameterised/parameterised neural networks respectively). We then have 4 sets of hidden layers. Each set consists of a 'standard' hidden layer with 200 units followed by a dropout layer to counteract overfitting. This dropout layer sets half the input values to zero randomly for each update during training time. This architecture is presented graphically in figure 23.

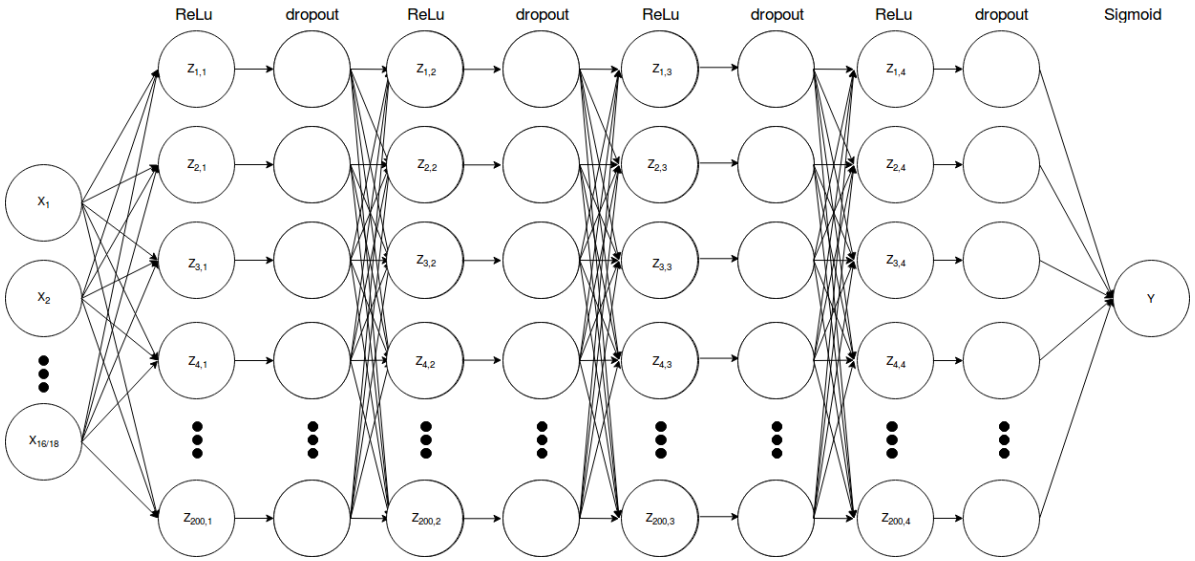
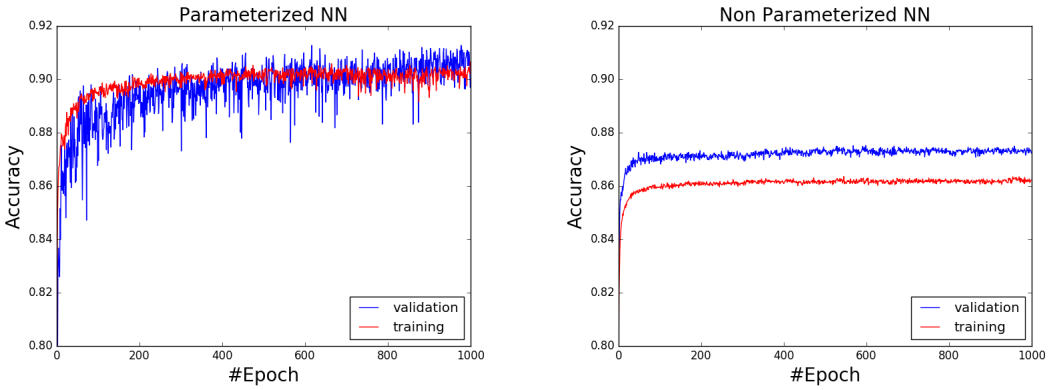


Figure 23: *visualisation of the network that will be used. We have 4 sets of hidden layers, each consisting of a 'normal' layer with 200 units and a dropout layer.*

5.2.2 Number of Epochs

Now we have an architecture set up, we only need to ask ourselves how many times data needs to be passed through before the weights are optimally set. This will be done by training a network up to a thousand epochs. This is the largest amount that still leads to reasonable computing times. After each epoch the accuracy of the training and validation sets will be measured. It must be noted that for the training and validation sets, accuracy is a good performance measure, as these sets are balanced, the amount of signal and background events are the same. Finally we will plot the accuracy versus the number of epochs to see when we can stop training. The results of this can be seen in figure 24.



(a) *Parameterised Neural Network.* (b) *Non-parameterised neural network.*

Figure 24: *The dependency of the accuracy of training and validation data on the number of epochs. The blue line indicates the validation data and the red line indicates the training data.*

We can see in figure 24 that 1000 epochs is enough for the weights to be optimally set. For the parameterised neural network, we see that the relation starts to level off around one hundred epochs. The training data levels off smoothly to an accuracy of just under 90%. The validation data follows the training data nicely. It is a bit more chaotic though. This could be because the network is not training on the validation set. Changes made to the weights that improve the performance of the training set might not improve the performance for the validation set. The fact that the accuracy of the validation set does not start to decrease after some number of epochs indicates that we do not have to worry about overfitting. This was to be expected because our networks include dropout layers. More or less the same things can be said about the non-parameterised NN. The relations start to level off a bit earlier, and at lower values. It makes sense that it starts to level off earlier, as it is trained on fewer features, and therefore less complex. It was also to be expected that it would have a worse performance, as literature tells us that the parameterised neural network should do a better job. The fact that the validation data does better here is a bit strange. The difference in accuracy is small though and it might be a coincidence that the validation data has more easily classifiable points. The same data was used to train and validate both networks, so we would expect the same thing for the parametrised case. We do see that the performance on the validation set still seems to be increasing. It could be that the parameterised network is not done improving after 1000 epochs, and that if we trained long enough, the validation performance will eventually show the same behaviour. We will later see another indication that the parameterized network is not fully trained yet.

We can conclude from these plots that cutting training at about 200 epochs would be reasonable. However, we have not encountered overfitting, and there is thus no harm in just using the models trained over a thousand epochs.

5.3 Results and Discussion

In the next couple of sections, we will present the results of the DNN. All results will be presented and discussed for both the parameterised and non-parameterised neural networks. We will first discuss the shape of the output distributions. We will then look at the performance of the networks from a machine learning perspective. Finally, we will look at the performance from a physical point of view.

5.3.1 Output distributions

The output distributions represent the distribution of the value Y in figure 23. All tested MSSM models give an output distribution with a similar shape. We will therefore discuss only one here. The output will be split in two. One group will have the output values of points that are (in reality) signal and the other of points that are (in reality) background. This way we can see how the network handles these differently. To get a clear picture, it is a good idea to look at a model where the number of signal points is not negligible compared to the amount of background points. We will later see that the performance of the network depends on the mass gap. We will therefore present a model here that has an average mass gap (40 GeV). Because of these two reasons we will choose the model point with $m_{\tilde{\chi}_2^0} = 106$ GeV and $m_{\tilde{\chi}_1^0} = 67$ GeV. The result is presented in figure 25.

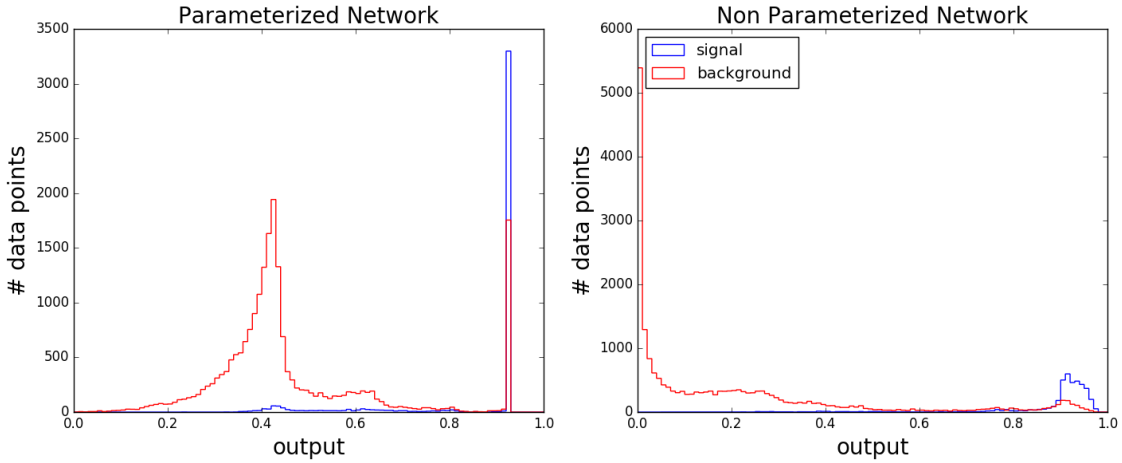


Figure 25: *The output distribution of the two networks. The parameterised NN is displayed on the left, the non-parameterised NN on the right. The red line indicates the output of points that are actually background and the blue points those that are actually signal. We can see that most background points are accumulated near lower outputs and signal points at higher outputs. For the parameterised NN, more than 90% of the background points have an output value <0.85. For the non-parameterised NN this is almost 95%.*

We can immediately see that both networks have separated the two classes well. For both networks, the background events are concentrated near low outputs, whereas signal events mostly result in high output values. There are some differences between the two networks. The parameterised network outputs a peak at a value just above 0.9. Almost all the signal is located here. A small fraction of the background is also located in this peak, but because there are so many background events, the amount of background events in this peak is still substantial. One might ask if the peak is not in fact two peaks, just very close together. This is not the case. Up to the precision of the computer, the values are all exactly the same. So, when choosing a threshold (the line that separates signal from background), you have to either include the peak or leave it out. But if you leave it out, you will classify no points as signal, so the only choice is leaving it in. This means that you have many true positives, but also a substantial amount of false positives, as there are also a lot of background events in the peak.

The non-parameterised network has roughly the same shape. However, it has a peak at almost zero, which consists exclusively of background points. There is also a peak at high values, but it is much more spread out. This means that when we choose a threshold, we have more room to choose one that gives the best TPR and FPR values. We will see later that the optimal

threshold will be somewhere in the middle of the peak. This means that you will also classify some signal points as background (false negatives). In the parameterised case, this almost didn't happen.

5.3.2 Performance

Now we have trained our network, we can test its performance. We will use two measures to determine the performance. As stated before, we cannot use accuracy as our test data is very imbalanced. We will therefore use the ROC-curves and AUC's per model point. A ROC-curve represents the trade-off between true positive rate and false positive rate. Classifying more points as positive might increase the true positive rate, but it will also increase the false positive rate. Good tools will have ROC-curves that come very close to FPR= 0 and TPR= 1. Figure 26 shows the ROC-curves of the different models. This plot is added to show that the shape of all ROC-curves is rather good, with all of them coming quite close to (0,1), although some do better than others. Comparing the results per model point is a bit hard from these plots. We will therefore focus on this when we discuss the AUC results.

One important thing that is already clear from these plots is that the non-parameterised neural network does a better job than the parameterised one. This is not what we expect. Literature (ref.[27]) and figure 24 both point in the opposite direction. One explanation could be that while the training seems to be completely done after 1000 epoch, this is not actually the case. Maybe the network has not learned the information presented by the mass parameters jet. It could be that this comes after more epochs, which would instigate a new rise in accuracy. Training so many epochs is very computationally intensive, so we advise future research with access to more computing resources to look into this.

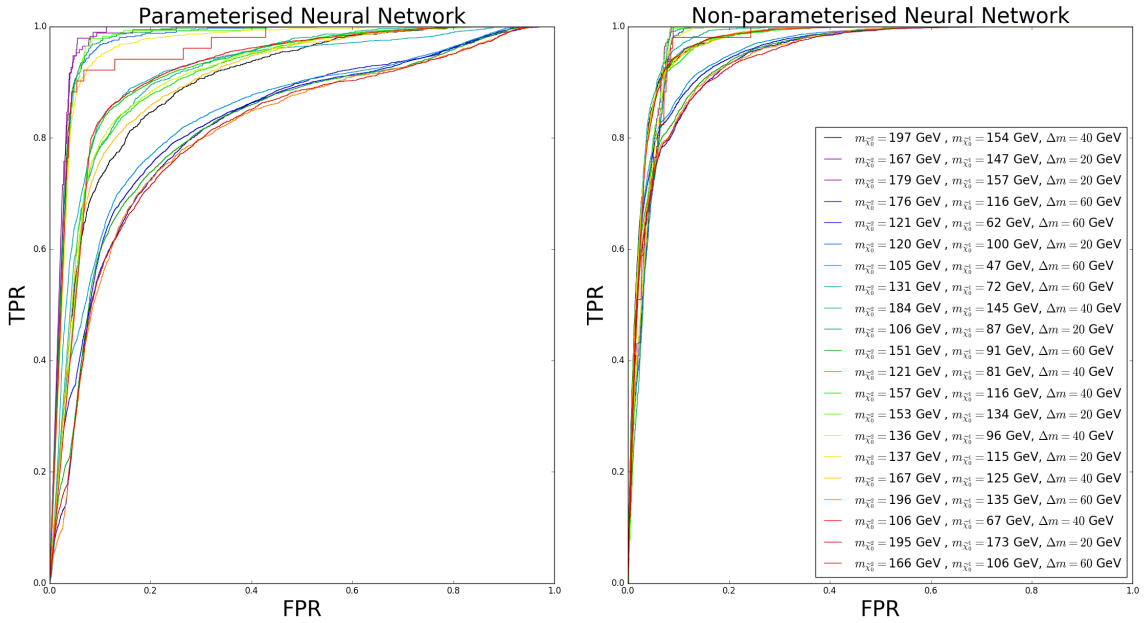


Figure 26: *The ROC-curves for the parameterised (left) and non-parameterised (right) neural network. Each line indicates a different model point tested on the same network. We can see that the non-parameterised neural network performs better.*

Next we will look at the performance of the network on the testdata by evaluating its AUC value. The AUC value (the area under the ROC-curve) is presented in figure 27. For the parameterised neural network, we can clearly see that the network does a better job at classi-

fying model points that have a small mass gap. This is a good thing, because we are trying to create a tool that works well on compressed models. We note that there is one point at about (130 GeV,70 GeV) that does not fit into this pattern. It is unclear why this is. It might be that this model-point is a bit different from the others in terms of other MSSM parameters not discussed in this thesis. However, we have tried to keep these parameters as constant as possible, so it is implausible that this causes such an outlier. The second observation is that the non-parameterised network does a better job. This NN also works better for models with smaller mass gaps, but the difference is much smaller.

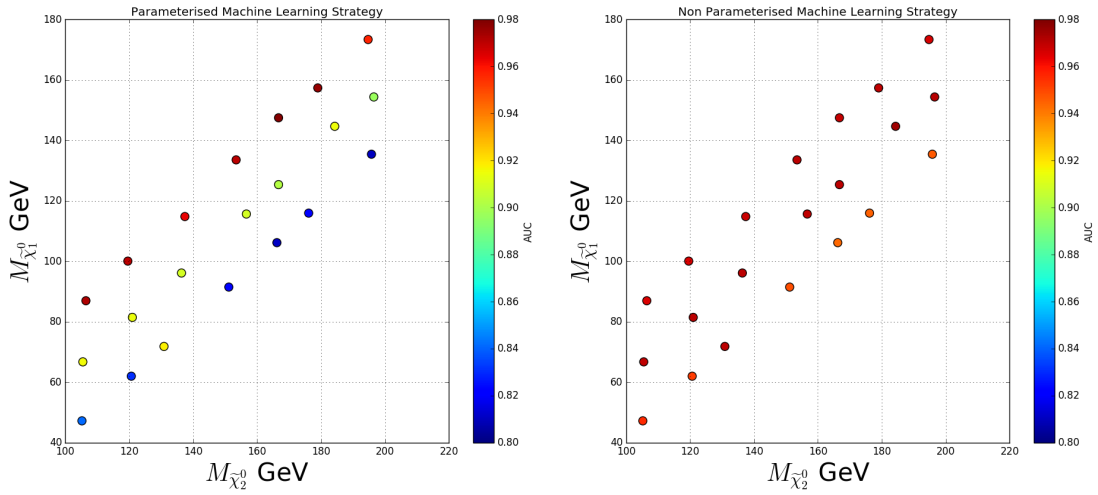


Figure 27: *The AUC values for the parameterised (left) and non-parameterised (right) neural network. We can see that while the non-parameterised neural network performs well for all models, the parameterised one is distinctly better at classifying models with ΔM small. We see a point that does not fit into the pattern in the left image, at about (130 GeV,70 GeV).*

5.3.3 Trustworthiness

As we have mentioned before, neural networks, especially large ones like we are using now, are black boxes. However, we can get some feel of what they are doing. We will do this by looking at the feature distributions (like figure 22) of the classes as the network has classified them and as we know them to be in reality. The network only gives an output, we still need to choose a threshold to get a classification. We will choose three different thresholds. By looking at how the feature distributions change with the thresholds, we can link the output to the features. We will take the same model as we chose for the output distributions and take thresholds 0.5, 0.7 and 0.9. We find that the only clear correlation occurs in the features $P_{T,l1}, P_{T,l2}, P_{T,l3}, P_{T,MET}$. We will look deeper into this by now taking a range of thresholds between 0.5 and 1, and making a colourplot where the y-axis represents the threshold, the x-axis the feature-value and the colour indicates the histogram value. We will discuss the results on the basis of the parameterised, signal output. The other plots give similar results and will thus not be discussed here.

We find that for all four features, a higher output means more points with a high P_T, although the relationship is less prominent for $P_{T,MET}$. At low thresholds, a large amount of points is classified as signal, and the distribution is quite spread out. When the threshold is increased, points with low output values are no longer classified as signal. At the same time, we see that the P_T distributions start to be more peaked around higher values. Apparently, a lower output value means a lower P_T.

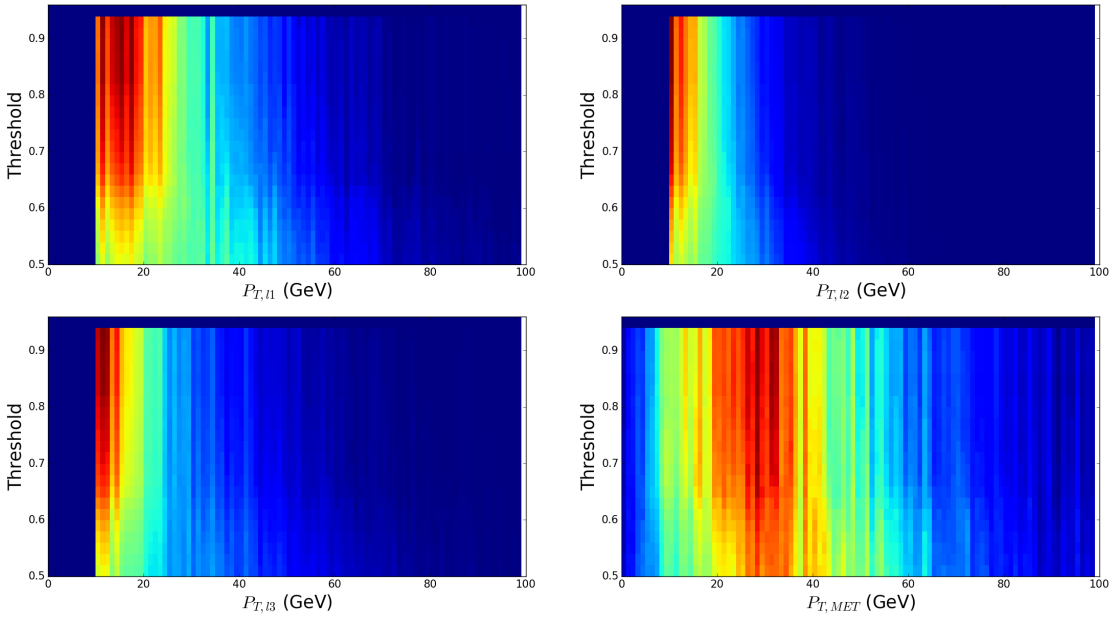


Figure 28: *The feature distribution of the classes as predicted by the parameterised neural network with different thresholds. A higher histogram value is indicated with a redder colour. We have not included a scale because the histograms are normalised, meaning that only relative differences are of importance. Here we compare the signal.*

5.3.4 Exclusion power

We can now safely say that the network has a good performance from a machine learning point of view. But how useful is it really? How does it stack up against a cut-based strategy [9]? We will assess this by looking at its exclusion power, the Z_N-value we introduced earlier. To calculate this, we need a classification. To find the optimal threshold, we will try 18 different thresholds from 0.88 to 0.97. These two values were selected by looking at the output plots. For each model point we will assess which threshold gives the highest significance. The results can be seen in figure 29.

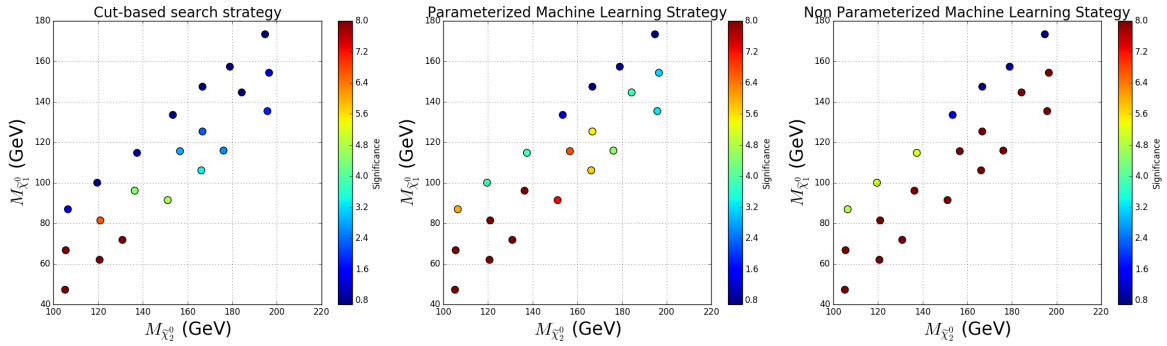


Figure 29: *The calculated significance per model for a strategy based on cuts (left), and for a strategy based on neural networks, parameterised (middle) and non-parameterised (right). A higher significance is better, and so we can see that the neural networks can improve substantially upon the cut-based method.*

We can clearly see that both networks result in significant improvement over the cut-based strategy. We can see that, just like with the cut-based strategy, the tool is more sensitive to models with low particle masses and high mass gaps. This can be explained by looking at

figure 21. The higher the signal fraction of a model (i.e. the higher the signal cross section) the more sensitive the tool is to it. This makes sense, there is relatively less background to 'hide' the signal in. The thing to notice is that both tools are sensitive to more models. They can work with lower signal fractions than the cut-based strategy can. This means that both tools also become sensitive to the very low mass gap models, which the cut-based strategy could not probe.

The significance plots can be seen as combinations of the performance plots (figure 27) and the signal fraction plots (figure 21). This explains why the ML tools are more sensitive to small mass gap models. Both tools, and especially the parameterised one, perform better at low mass gaps. This (partly) counteracts the fact that these models also have very small signal fractions, which would normally mean that we are less sensitive to them. We can also see that again the non-parameterised tool does a much better job than the parameterised tool. The non-parameterised tool is such an improvement over the cut-based strategy, that we cannot no longer learn from this selection of SUSY models at what masses the tool loses sensitivity (Significance $< 2\sigma$). We expect the sensitivity of the plot to go down at higher masses, but we have not tested high enough masses to know when this occurs. Future research should look into this. The parameterised tool starts to lose sensitivity at an LSP mass of $\sim$150 GeV for small mass gaps and $\sim$200 GeV for larger mass gaps.

6 Conclusion and outlook

We will now discuss the most important conclusion from both projects and give an outlook for the future. From our first project it is clear that non-linear classification tools like support vector machines and neural networks can greatly improve the classification of lepton pairs. The tools used were all relatively simple, and easily applicable. It is therefore our recommendation that future research looks into using these types of tools for these types of classification problems. Our results were qualitative. It would be interesting to see if they could be made quantitative by not looking at hard scattering events but at realistic detector events. Opening the black box would also be extremely useful, as this would give us new parameters with which to classify.

In our second project we have proven that ML tools can be used to classify signal and background processes in compressed miniminal supersymmetric dark matter search strategies. Future research should expand on this, by looking at more models and seeing how far the performance can be enhanced by tuning the network architecture. Another question that remains is why the parameterised tool does worse than the non-parameterised one. This should be addressed by training larger networks that can actually make use of the theory parameter features. Another important goal for future research should be to repeat this research with all background processes included, not only the irreducible ones. Can machine learning then still improve on the cut-based strategies?

All in all, we have shown that ML tools can be used for classification within events as well as among events. In particle physics, ML remains a relatively unexplored field, and it is our hope that in the upcoming years this will change, and that the realization that ML can greatly improve our sensitivity to new-physics signals will become more wide-spread.

References

[1] K. Freese, *Status of dark matter in the universe*, *International Journal of Modern Physics* **D26** (2017) .

[2] K. G. Begeman, A. H. Broeils and R. H. Sanders, *Extended rotation curves of spiral galaxies: dark haloes and modified dynamics*, *Monthly Notices of the Royal Astronomical Society* **249** (1991) 523–537.

[3] G. Bertone, D. Hooper and J. Silk, *Particle dark matter: evidence, candidates and constraints*, *Physics Reports* **405** (2005) 279–390.

[4] N. A. Bahcall, *Clusters and superclusters of galaxies*, in *Formation of structure in the universe*, no. 1958, p. 50, 1996.

[5] J. L. Feng, *Dark Matter Candidates from Particle Physics and Methods of Detection*, *Annual Review of Astronomy and Astrophysics* **48** (2010) 495.

[6] D. Griffiths, *Introduction to Elementary Particles*. Wiley-VCH, 2nd ed., 2008.

[7] S. Caron, A. Achterberg, L. Hendriks, R. Ruiz de Austri and C. Weniger, *A description of the Galactic Center excess in the Minimal Supersymmetric Standard Model*, *JCAP* **08** (2015) .

[8] G. Bertone et al., *Global analysis of the pMSSM in light of the Fermi GeV excess: prospects for the LHC Run-II and astroparticle experiments*, *JCAP* **4** (2016) 37.

[9] M. van Beekveld, W. Beenakker, S. Caron and R. R. de Austri, *The case for 100 GeV bino dark matter: a dedicated LHC tri-lepton search*, *JHEP* **04** (2016) .

[10] ATLAS collaboration, *Search for electroweak production of supersymmetric particles in the two and three lepton final state at* $\sqrt{s} = 13$ *TeV with the ATLAS detector*, Tech. Rep. ATLAS-CONF-2017-039, CERN, Geneva, Jun, 2017.

[11] F. Pedregosa et al., *Scikit-learn: Machine Learning in Python*, *Journal of Machine Learning Research* **12** (2011) 2825.

[12] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[13] J. F. Trevor Hastie, Robert Tibshirani, *The Elements of Statistical Learning*. Springer, 2nd ed., 2008.

[14] I. H. Witten, E. Frank and M. A. Hall, *Data mining*. Elsevier Science and Technology, 2nd ed., 2005.

[15] T. M. Mitchell, *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.

[16] A. Damien et al., "Tflearn." https://github.com/tflearn/tflearn, 2016.

[17] M. Abadi et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, `[arXiv:1603:04467]`.

[18] F. Chollet and Others, *Keras*, 2015.

[19] J. Alwall, M. Herquet, F. Maltoni, O. Mattelaerc and T. Stelzerd, *MadGraph 5: Going beyond*, *JHEP* **06** (2011) .

[20] T. Sjöstrand, S. Mrenna and P. Skands, *PYTHIA 6.4 physics and manual*, *JHEP* (2006) 26.

[21] M. Cacciari, G. P. Salam and G. Soyez, *FastJet user manual*, *The European Physical Journal* **C72** (2011) .

[22] *Dispelling the n3 myth for the kt jet-finder*, *Physics Letters* **B641** (2006) 57.

[23] M. van Beekveld, *Possible Indirect Detection of Dark Matter and its impact on LHC Supersymmetry searches*, Master's thesis, Radboud Universeit Nijmegen, the Netherlands, 2016.

[24] J. De Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens et al., *DELPHES 3: A modular framework for fast simulation of a generic collider experiment*, *JHEP* **2** (2014) .

[25] E. Conte, B. Fuks and G. Serret, *MadAnalysis 5, a user-friendly framework for collider phenomenology*, *Computer Physics Communications* **184** (2013) 222.

[26] P. Baldi, P. Sadowski and D. Whiteson, *Searching for exotic particles in high-energy physics with deep learning*, *Nature Communications* **4308** (2014) .

[27] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski and D. Whiteson, *Parameterized neural networks for high-energy physics*, *European Physical Journal C* **76** (2016) .