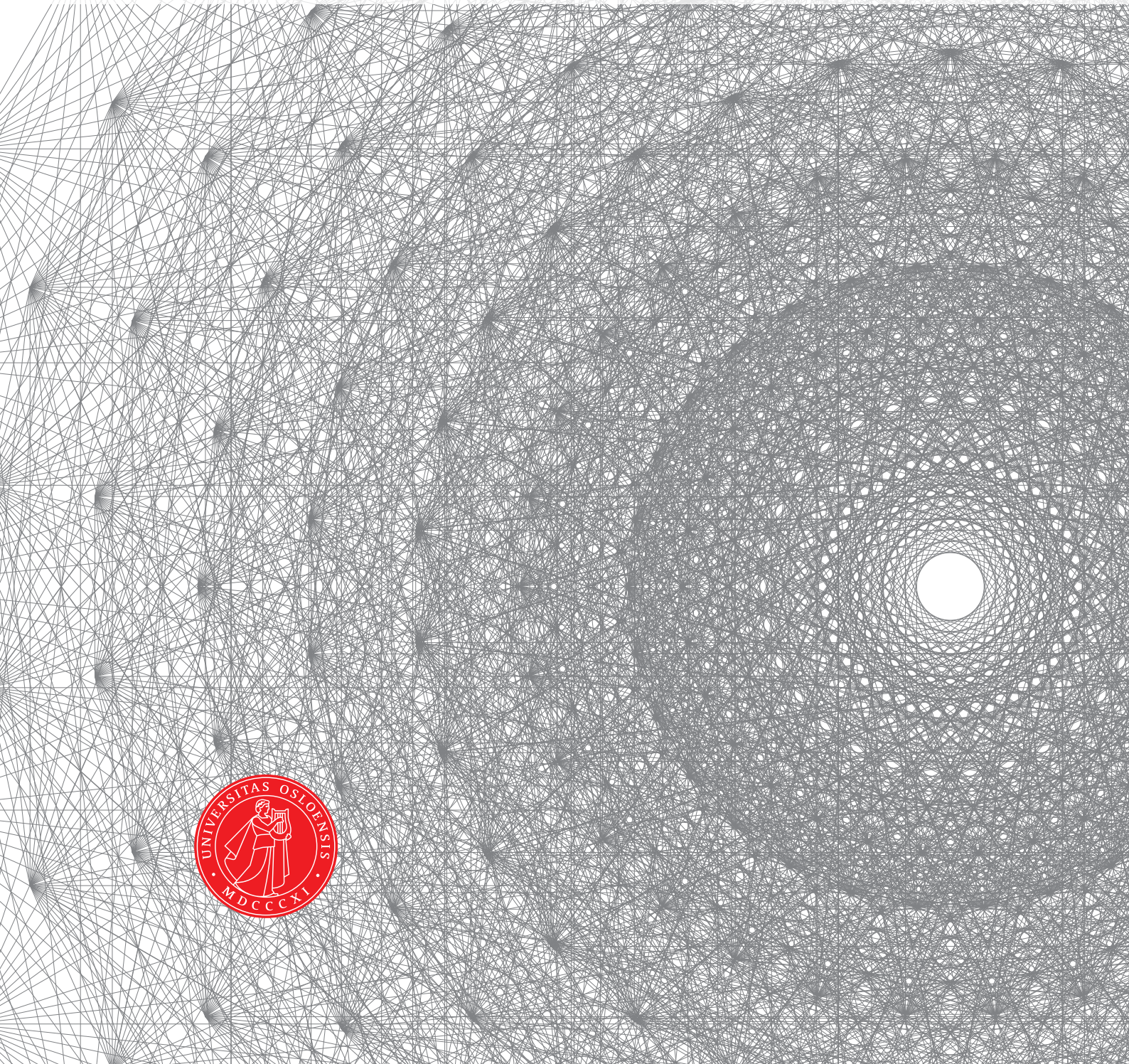
UiO : Department of Physics
University of Oslo

Fast Evaluation of Supersymmetric Cross Sections

Jon Vegard Sparre

Master's Thesis, Spring 2018



This master's thesis is submitted under the master's programme *Physics*, with programme option *Theoretical Physics*, at the Department of Physics, University of Oslo. The scope of the thesis is 60 credits.

The front page depicts a section of the root system of the exceptional Lie group E_8, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of physics.

Samandrag

I denne masteroppgåva presenterer vi ein rask og presis metode for utrekning av nestleiande ordens supersymmetriske tverrsnitt. Metoden går ut på å trene maskinlæringsalgoritmen forsterska avgjerdstre på eit generert datasett med datapunkt frå parameterrommet til MSSM-24 og dei tilhøyrande NLO-tverrsnitta for ein gitt prosess utrekna med `Prospino 2.1`. For å vise at metoden fungerer, nyttar vi gluino parproduksjon frå ein proton–proton starttilstand som døme. Vi vil sjå at den resulterande avgjerdstremodellen vil ha reknefeil mykje mindre enn andre reknefeil frå andre feilkjelder som partontettleiksfunksjonane, α_s og bidrag frå høgare ordenar. Vi trur at denne metoden enkelt kan generaliserast til andre supersymmetriske prosessar og dimed vere veldig nyttig for å gjere kraftigare globale tilpassingar.

Abstract

In this thesis we present a fast and precise method for evaluating supersymmetric cross sections at next-to-leading order (NLO). The method consists of training the machine learning algorithm boosted desicion trees on a generated data set with samples from the parameter space of the MSSM-24 and their corresponding NLO cross sections for a given process, calculated with `Prospino 2.1`. We use gluino pair production with proton-proton initial state as an example to show the viability of the method. The resulting predictive model has errors well below other error sources such as the parton density functions, α_s, and contributions from higher orders. We believe that this method can be easily generalized to other supersymmetric processes and thus very useful for extending the power of global fits.

Takk

Eg takkar professor Are Raklev for god rettleiing i arbeidet med denne masteroppgåva, det har vore både artig og frustrerande, og eg har lært meg å sette pris på klarspråket ditt i tilbakemeldingane eg har fått. Takk til Anders for `point_sampler.py`. Eg vil også takke medstudentane og etter kvart vennane som har vore til hjelp i studietida, mest av alt sosialt. Spesielt vil eg takke Anne-Marthe, Henrik, og Vigdis for at de er så kjekke og greie.

Til slutt vil eg takke Stian. Eg var heldig som traff deg da masterstudiet tok til og ikkje seinare. Du er god å ha.

Dette arbeidet har vore støtta av Lånekassa.

The CPU intensive part of this work was performed on the Abel Cluster, owned by the University of Oslo and the Norwegian metacenter for High Performance Computing (NOTUR), and operated by the Research Computing Services group at USIT, the University of Oslo IT-department. The computing time was given by NOTUR allocation NN9284K, financed through the Research Council of Norway.

Blindern, 1. februar 2018
Jon Vegard Sparre

Contents

Introduction

Over the last century physicists have made a large effort to build a theoretical model that describes the most fundamental interactions and particles in Nature, the *Standard Model* (SM). It has proven to be a very successful model with some of the best agreements between theory and experiment ever achieved. However, the Standard Model has some shortcomings. One of them is the *hierarchy problem* introduced by the *Higgs mechanism*, which requires extremely fine tuning of the theory in order to make it work. Physicists tend to like theories that are natural, *i.e.* theories without the need of fine tuning. This is one of the reasons that the Standard Model is not believed to be the most fundamental theory of Nature.

A possible extension of the SM is *supersymmetry* (SUSY) which is a symmetry between fermions and bosons. In supersymmetry all SM fermions have a bosonic supersymmetric partner and vice versa. The new *sparticles* in supersymmetry thus differ from their SM partners by half a unit of spin. Supersymmetry solves several of the problems with the SM, such as the hierarchy problem and it provides multiple dark matter candidates.

One of the goals of the Large Hadron Collider experiment at the research facility CERN is to discover supersymmetry. It has been succesful in discovering the Higgs boson, now everyone waits for the discovery of supersymmetry. The experiment has been running since 2008 and have collected proton-proton collision data at 7 TeV, 8 TeV and currently at 13 TeV center-of-mass energy. When searching for supersymmetry one approach is to do global fits and find what supersymmetry can and cannot be, *i.e.* see which parts of the parameter space for supersymmetric models are excluded. To do global fits and exclusions it is crucial to know the supersymmetric production cross sections at the highest accuracy possible.

In this thesis we will develop a faster and more general predictive model than currently available for supersymmetric production cross sections at next-to-leading order by using machine learning. The method used in this thesis should in principle be useful for all production cross sections for any physical models given enough data to train with. We will use the cross section for gluino pair production from proton-proton collisions, $\sigma(pp \to \tilde{g}\tilde{g})$, as an example to show that this method is viable. State-of-the-art methods for calculating supersymmetric cross sections at next-to-leading order relies on `Prospino 2.1` [1, 2] which takes $\sim$ 15 minutes for each parameter point, and `NLL-fast 2.1` [3, 4, 5, 6] which uses a few seconds for each parameter point and also gives us the next-to-leading-logarithmic correction, but is restricted by the assumption of degenerate squark masses. Both of the programs are therefore limited in use.

The data used for training the machine learning algorithm were generated

in large amounts on the Abel computer cluster. The training data was sampled in two different ways to ensure good coverage of the parameter space of the MSSM-24. The data have also been quality checked to remove outliers and pre-processed in order to make it easier for the machine learning algorithm to learn the next-to-leading order cross section function. We have used the algorithm *gradient boosted decision trees* in this thesis. This is an algorithm robust of overfitting to the data and which is easy to interpret. The resulting predictive models have been tested in several ways, both with built-in methods in the program package and self-defined methods. We have also compared our method against `NLL-fast 2.1`.

The thesis is structured as follows. We will introduce basic supersymmetry concepts and phenomenology in Chapter 1, then we will study the analytical leading-order cross sections for gluino pair production and how the next-to-leading order cross section is calculated by today's methods in Chapter 2. In Chapter 3 we will delve into the world of machine learning and discuss some of the techniques available, with main focus on the boosted decision tree algorithm, and we will also discuss how to measure the performance of machine learning models. In Chapter 4 we present and discuss the details of the model building, while in Chapter 5 we present and discuss our results before making our conclusions.

Chapter 1

The Standard Model and Supersymmetry

In this chapter we will briefly describe the Standard Model and its limitations before introducing supersymmetry. We will focus on the Minimal Supersymmetric Standard Model.

1.1 The Standard Model

We will recap some important points from the Standard Model (SM), before introducing the symmetry groups used in the SM. Then we will introduce the Higgs mechanism before ending with a brief discussion of the Standard Model's limitations as a primer for the section on supersymmetry. We will assume knowledge of quantum field theories and necessary group theory basics.

1.1.1 Field Theory Basics

The Standard Model is a highly successful theory describing the most fundamental physcial interactions known to us. It is a *quantum field theory* (QFT) which means that particles and their interactions are described by fields. Quantum field theory is a mixture of the successful *quantum mechanics* and *special relativity* theories. The former describes the physics of the smallest scales, while the latter describes things that move fast.[1] By combining these we get a theory for describing small things moving fast, which is very appropiate for physics at the Large Hadron Collider.

A quantum field theory is based on the *Lagrangian density* $\mathcal{L}$ which is a function of fields $\psi_i(x)$ and their derivatives $\partial_\mu \psi_i(x)$. The equations of motion of the fields are obtained via the Euler-Lagrange equation,

$$\frac{\partial \mathcal{L}}{\partial \psi_i(x)} - \partial_\mu \left(\frac{\partial \mathcal{L}}{\partial(\partial_\mu \psi_i(x))} \right) = 0, \tag{1.1}$$

where $i = 1, \ldots, N$ is the index for each field. So far this description is also valid

[1] For the record: Special relativity also describes slow things.

for classical field theories.[2] The Lagrangian density is related to the *action* S by

$$S = \int \mathrm{d}^4x\, \mathcal{L}, \tag{1.2}$$

where x is the four dimensional spacetime, and the Euler-Lagrange equation can be found from the principle of least action.

1.1.2 Symmetries

One of the most important theorems in physics is *Noethers theorem* [7], it states that *every differentiable symmetry of the action of a physical problem has a corresponding conservation law*. Thus, if we can continuously transform $\mathcal{L}$ and leave the equations of motion invariant, we say that we have a symmetry leading to a corresponding conserved quantity. We distinguish between *external* and *internal* symmetries. In the SM the former is invariance of the Lagrangian under spacetime symmetries.[3] The latter is invariance under transformations of the fields themselves.

To describe the internal symmetries we rely on group theory. It is convenient to find field transformations using properties of groups, for example using $SU(n)$ and $U(n)$, that describe continuous symmetries.[4] If we have a field $\psi(x)$ it transform as,

$$\psi(x) \to \psi'(x) = e^{-ig\alpha^a(x)T^a}\psi(x), \tag{1.3}$$

where $\alpha^a(x)$ are the transformation parameters,[5] and T^a the *generators* of the group. The generators of a group are the group elements that can produce all other group elements. We have as many transformation parameters as there are generators of the group.

Since $\mathcal{L}$ also depends on the derivatives of the fields they must transform as in Eq. (1.3). Naively one may try with the pure derivative $\partial_\mu\psi(x)$, however, due to the locality of $\alpha^a(x)$ we have to introduce the covariant derivative,

$$D_\mu = \partial_\mu + igA^a_\mu(x)T^a, \tag{1.4}$$

where $A^a_\mu(x)$ are *gauge boson fields* introduced in order to keep $\mathcal{L}$ invariant. The A^a_μ must transform according to,

$$A^a_\mu(x) \to A'^a_\mu(x) = A^a_\mu(x) + \partial_\mu\alpha^a(x) + gf^a_{bc}\alpha^b(x)A^c_\mu, \tag{1.5}$$

where g is the coupling constant of the gauge fields, and f^{abc} are the structure constants of the gauge group. Thus $D_\mu\psi(x)$ will leave the Lagrangian invariant. For the $U(1)$ gauge group, an Abelian group, the structure constants are zero. We see from Eq. (1.4) that we will get as many gauge boson fields as there are generators of the group. The Standard Model is based on the $SU(3)_C \times SU(2)_L \times U(1)_Y$ symmetries, where the former describes quantum chromodynamics and

[2] The quantum part comes into play when we introduce creation and annihilation *operators*, which we expand the fields in terms of. A particle is actually an excited state of a field.

[3] Example: Conserved quantities due to spacetime symmetries such as translations in time and space are energy and momentum, respectively.

[4] $SU(n)$ is the set of all complex valued and unitary $n \times n$ matrices with determinant 1. $U(n)$ is defined in the same way as $SU(n)$ without the requirement on the determinant.

[5] It can be anything, really. (Which means that it must be a real and differentiable function which depends on spacetime.)

the two latter the electroweak interaction. $SU(3)_C$ and $SU(2)_L \times U(1)_Y$ gives us eight and four gauge boson fields respectively.

Thus, by requiring certain symmetry transformations and using properties of suitable groups we find the complete covariant derivative for the Standard Model,

$$D_\mu = \partial_\mu + ig_s \frac{\lambda^a}{2} C_\mu^a(x) + ig \frac{\sigma^a}{2} W_\mu^a(x) + \frac{i}{2} g' Y B_\mu(x), \tag{1.6}$$

where $C_\mu^a(x)$, and $W_\mu^a(x)$ and $B_\mu(x)$ are the massless gauge fields for the strong and electroweak interactions, and g_s, g, and g' are the corresponding coupling constants. The generators of $SU(3)_C$ are 1/2 times the Gell-Mann matrices λ^a, while the generators of $SU(2)_L$ are the Pauli matrices σ^a. Due to electroweak symmetry breaking the $W_\mu^a(x)$ and $B_\mu(x)$ are not the physical gauge boson fields, they are instead mixed into four linear combinations that are the physical fields $W^\pm$, Z^0, and γ that we observe, and where $W^\pm$ and Z^0 become massive from absorbing Higgs field components due to the *Higgs mechanism*, see next section.

The physical quantities conserved for each group symmetry is the color charge C under $SU(3)_C$, weak isospin I_3 under $SU(2)_L$, and weak hyper charge[6] Y under $U(1)_Y$.

Thus, by using the relatively simple[7] principle of symmetry, we can develop the framework of the most fundamental physical interactions.

1.1.3 Higgs Mechanism

To give a more complete picture of the Standard Model we will also briefly introduce the Higgs mechanism. One of the early issues with the Standard Model was the introduction of the electroweak bosons $W^\pm$ and Z^0. They did not have mass terms in the SM Lagrangian, but we knew they had to be massive, otherwise they would have been observed a long time ago. In order to give them masses the mechanism found independently by Guralnik, Hagen, and Kipple [8], Brout and Englert [9], and Higgs [10], later known as the Higgs mechanism, was introduced.

The mechanism involves an $SU(2)$ doublet we call the Higgs doublet $\Phi(x)$ giving us new terms in the Lagrangian,

$$\mathcal{L} \supset |D_\mu \Phi(x)|^2 - \mu^2 |\Phi(x)|^2 - \lambda |\Phi(x)|^4, \tag{1.7}$$

where the covariant derivative is,

$$D_\mu = \partial_\mu + ig \frac{\sigma^a}{2} W_\mu^a(x) + \frac{ig'}{2} Y B_\mu(x). \tag{1.8}$$

The last two terms in Eq. (1.7) are the constituents of the *Higgs potential* with $\mu^2, \lambda \in \mathbb{R}$. The hypercharge of the Higgs field is $Y = 1$.

It is the vacuum state, $\Phi_0 = (\phi_{a0}, \phi_{b0})^T$, of the Higgs field that turns out to be the interesting part. It will give us the electroweak symmetry breaking and give the $W^\pm$ and Z^0 their masses. The vacuum state corresponds to the minimum of the Higgs potential. In order to have a stable vacuum state we

[6] Weak hyper charge is related to electric charge Q via $Y = 2(Q - I_3)$.

[7] Maybe not simple mathematics, though.

must require $\lambda > 0$ such that the potential is bounded from below.[8] Further on we want the minimum to give us a non-zero *vacuum expectation value* of the Higgs field such that the symmetry of the potential (and thus the electroweak symmetry) is broken. To achieve this we require $\mu^2 < 0$, which gives us the well known wine bottle potential.[9] We may then write,

$$|\Phi_0|^2 = |\phi_{a0}|^2 + |\phi_{b0}|^2 = \frac{-\mu^2}{2\lambda} \equiv \frac{v^2}{2}, \tag{1.9}$$

where we have introduced the v as the vacuum expectation value. The physical vacuum state will then correspond to a point on the circle, breaking the electroweak symmetry spontaneously, we call this *spontaneous symmetry breaking*.

We can now write the vacuum state as,

$$\Phi_0(x) = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ v + h(x) \end{pmatrix}, \tag{1.10}$$

without loss of generality.[10] In order to find the particle states we perturb the field around the vacuum state,

$$\Phi(x) = \frac{1}{\sqrt{2}} \begin{pmatrix} \eta_1(x) + i\eta_2(x) \\ v + h(x) + i\eta_3(x) \end{pmatrix}, \tag{1.11}$$

where the perturbations $\eta_i(x)$ and $h(x)$ are four real scalar fields. However, the $\eta_i(x)$ gives us a lot of new terms in Eq. (1.7) which are called *Goldstone bosons*. Luckily we are working with a gauge invariant theory, thus we are allowed to do a gauge transformation such that these fields will be "eaten" by the $W^{\pm}$ and Z^0. The so-called *unitary gauge* gives us,

$$\Phi(x) \to \Phi'(x) = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ v + h(x) \end{pmatrix}. \tag{1.12}$$

We may now write Eq. (1.7) in the unitary gauge,[11]

$$\mathcal{L} \supset \frac{v^2 g^2}{8}(W^1_\mu W^{1\mu} + W^2_\mu W^{2\mu}) + \frac{v^2}{8}(gW^3_\mu - g'B_\mu)(gW^{3\mu} - g'B^\mu). \tag{1.13}$$

We observe that the three Goldstone bosons were "eaten" by the electroweak gauge bosons, thus giving them one more degree of freedom: their masses. The gauge states are mixed, giving the physical states,

$$W^{\pm}_\mu = \frac{1}{\sqrt{2}}(W^1\mu \mp iW^2_\mu), \quad Z^0_\mu = \frac{gW^3_\mu - g'B_\mu}{\sqrt{g^2 + g'^2}},$$
$$A_\mu = \frac{g'W^3_\mu - gB_\mu}{\sqrt{g^2 + g'^2}},$$

where the $A_\mu(x)$ is the photon field. Their respective masses are,

$$m_W = \frac{1}{2}gv, \quad m_Z = \frac{1}{2}\sqrt{g^2 + g'^2}v, \quad m_A = 0. \tag{1.14}$$

[8] If the potential is not bounded from below there is a probability to have tunneling out of the minimum giving us infinite negative energy (which ruins everything).

[9] Cheers!

[10] Since the potential is only dependent on $|\Phi|^2$.

[11] Also the W^a_μ and B_μ must be transformed according to their transformation properties.

One $U(1)$ symmetry is left in the Lagrangian after expansion around v since the photon is massless, the conserved quantity is the electric charge.

We also note that the same mechanism gives rise to the fermion masses, without going into details here.

1.1.4 Limitations of the Standard Model

With some group magic and symmetries we develop the framework describing the most fundamental physical interactions. The last 60 years has seen great discoveries in physics by using the Standard Model and its predecessors. An example is the calculation of the fine-structure constant α which is calculated in [11], by using measurements of the dimensionless magnetic moment of the electron g [12] and Feynman-diagrams up to eighth order, to be $\alpha^{-1} = 137.035\ 999\ 070(98)$, which is a strikingly precise determination of physical quantity.

However, the Standard Model has some limitations. Maybe the most obvious is the absence of gravity, which makes the Standard Model an incomplete theory in the sense that it does not describe everything.

Another issue with the Standard Model is that it does not have any viable candidate for dark matter. The past $\sim$ 100 years of observations have given us strong evidence of the existence of dark matter as a new unknown particle, and measurements have shown that as much as 26% of the Universe consists of it [13]. The new unknown particle is believed to be a weakly interacting massive particle (WIMP), *i.e.* the particle should only interact via the weak and gravitational forces with Standard Model particles. The only Standard Model particles fulfilling this requirement are the neutrinos, but they are too light with their mass of at most a few electronvolts.[12]

The last problem in this incomplete summary is the *hierarchy problem*. In short it is a problem due to quantum corretions in the Higgs mass from loop diagrams leading to a huge theoretical overestimate of the Higgs boson mass compated to experimental data. It will be discussed more thoroughly in Section 1.2.1.

With these considerations in mind we may think of improvements of the existing theory. Luckily people have been doing that, and in the following section we will discuss one proposed extension of the Standard Model, namely *supersymmetry*.

1.2 Supersymmetry

Now we will introduce the most important concepts of supersymmetry. We start with the basic terminology and motivation, and then we will look at the *Minimal Supersymmetric Standard Model* (MSSM) and discuss its content. We will briefly mention the *Constrained Minimal Supersymmetric Standard Model* as a realization of the MSSM, then discuss the soft supersymmetric Lagrangian before closing the chapter with some MSSM phenomenology. A more thorough introduction and discussion of the consequences of supersymmetry can be found in [14] and [15].

[12] The neutrinos' low mass is also something the Standard Model has problems to describe as there are no mass terms for neutrinos in the original formulation.

The basic idea behind Supersymmetry is quite simple. We extend the Standard Model by introducing a symmetry between fermions and bosons,

$$Q|\text{fermion}\rangle = |\text{boson}\rangle, \qquad Q|\text{boson}\rangle = |\text{fermion}\rangle, \tag{1.15}$$

where Q is an operator that transforms the spin of a state by a factor $1/2$. In Table 1.1 we have listed the different types of SM particles and their supersymmetric partners. We call the supersymmetric particles *sparticles*. Supersymmetric partners of SM fermions are named with an additional "s-" as a prefix, *e.g.* the supersymmetric partner of the electron becomes the *selectron*. While for the supersymmetric partners of SM bosons we add the suffix "-ino", *e.g.* the SM particle gluon has the supersymmetric partner *gluino*. When writing the sparticles symbolically we use a tilde above the particle's symbol, the gluino is thus written $\tilde{g}$.

SM particle (spin)	SUSY particle (spin)
Fermion (1/2)	Scalar (0)
Vector boson (1)	Fermion (1/2)
Scalar boson (0)	Fermion (1/2)

Table 1.1: Overview over the types of Standard Model particles and their supersymmetric partners.

If supersymmetry had been an exact symmetry, the masses of particles and their corresponding sparticles should be equal. This is obviously not the case, otherwise we would have discovered sparticles a long time ago. Thus we know that supersymmetry must be a broken symmetry. The question then is how heavy the sparticles may be. To give an estimate for this we may consider the *hierarchy problem*.

1.2.1 Hierarchy Problem

The hierarchy problem is a problem that arises when one tries to calculate the loop-corrections to the mass of the Higgs boson. We can write the physical mass as,

$$m_h^2 = (m_h^0)^2 + \Delta m_h^2, \tag{1.16}$$

where m_h^0 is the Lagrangian Higgs mass and Δm_h is the corrections due to loop diagrams, such as those in Fig. 1.1.

When calculating loop diagrams one gets divergent integrals. To deal with this one does *regularization*,[13] which is to introduce a cut-off scale such that we limit the momentum in the loop integrals to below a scale Λ_{UV}. The most natural choice of this scale is based on that the Standard Model is an incomplete theory. When we go up to Planck scale energies we expect new physics, such as quantum gravity. We can therefore set $\Lambda_{\text{UV}} \sim 10^{18}$ GeV.

The fermion and scalar loop corrections to the Higgs mass will at leading order then be,

$$\Delta m_h^2 = -\frac{|\lambda_f|^2}{8\pi^2}\Lambda_{\text{UV}}^2 + \frac{\lambda_s}{8\pi^2}\Lambda_{\text{UV}}^2 + \dots, \tag{1.17}$$

[13] There are several methods, *e.g. dimensional regularization*.

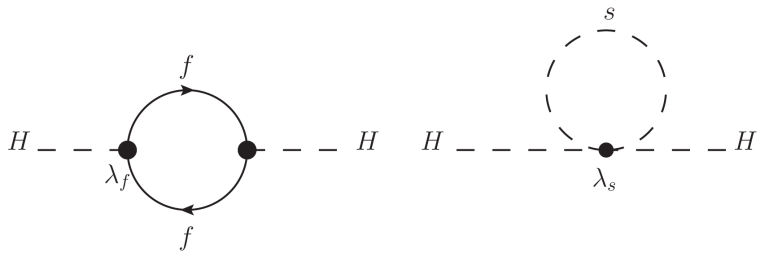


Figure 1.1: One loop contributions to the mass of the Higgs H from fermions f (left) and scalars s (right) loop. Figure taken from [15].

where λ_f is the fermion coupling to the Higgs, and λ_s the scalar coupling to the Higgs. This will lead to huge contributions of the order Λ_{UV} which is not compatible with the observed Higgs mass at ~ 125 GeV. Thus there must be a enormous cancellation in the corrections requiring that the terms with different signs are extremely fine tuned.

However, in supersymmetry it turns out that the divergences of the Higgs mass cancel. In unbroken supersymmetry one has $|\lambda_f|^2 = \lambda_s$ and exactly twice as many scalar degrees of freedom as fermion degrees of freedom running around in loops, which gives us this magic cancellation between scalar and fermion loops. This is one of the main motivations for supersymmetry.

Still, this must also be true for the broken supersymmetry if it should function as a solution to the hierarchy problem. To maintain the relationships between the dimensionless couplings we are led to *soft* supersymmetry breaking which does not (re)introduce the quadratic divergences. We can write the effective Lagrangian of the MSSM as,

$$\mathcal{L} = \mathcal{L}_{\text{SUSY-SM}} + \mathcal{L}_{\text{soft}}, \tag{1.18}$$

where the first term contains all gauge and Yukawa interactions and preserves supersymmetry, while the latter violates supersymmetry but contains only mass terms and coupling parameters of *positive mass dimension*.

The leading corrections to the Higgs mass are now,

$$\Delta m_h^2 = -\frac{\lambda_s}{16\pi^2} m_s^2 \ln\left(\frac{\Lambda_{\text{UV}}^2}{m_s^2}\right), \tag{1.19}$$

where m_s is the mass scale of the soft breaking terms. To keep the corrections small enough not to reintroduce the hierarchy problem we must have $m_s \sim \mathcal{O}(1 \text{ TeV})$. Thus we expect new physics at the TeV scale. The absence of these new particles is often called the *little hierachy problem*.

1.2.2 R-parity

In supersymmetry there is no automatic conservation of lepton and baryon numbers since the only requirements are gauge and supersymmetry invariance. Thus we arrive at predictions like rapid proton decay. However, experimental constraints on proton decay sets a lower limit of the life time of protons at $\tau > 5.9 \cdot 10^{33}$ years [16].

To avoid such predictions a possible solution is to introduce a new, discrete symmetry called *R-parity*. It is defined as,

$$P_R = (-1)^{3(B-L)+2s}, \tag{1.20}$$

where B is the baryon number, L lepton number and s particle spin. This leads to $P_R = -1$ for sparticles and $P_R = +1$ for particles and Higgs bosons. The symmetry is implemented in the Lagrangian by only allowing interaction terms obeying R-parity.

The phenomenological consequences of this symmetry are important: When producing sparticles from particles one always produce sparticle *pairs*. Sparticles will also always decay into an odd number of lighter sparticles and the lightest sparticle will be stable since it has nothing to decay to.

Since we will have a stable *lightest supersymmetric particle* (LSP) we know that it must be invisible to a detector, if not we should have found it already, for example in cosmic rays. The LSP thus interacts at most weakly and must be color and electrically neutral. This makes it a good candidate for dark matter. A neutralino (see Table 1.2) is often considered to be the LSP since it is the most suitable dark matter candidate.[14]

1.2.3 The Minimal Supersymmetric Standard Model

The most popular supersymmetric extensions of the Standard Model is the *Minimal Supersymmetric Standard Model* (MSSM). This is the minimal supersymmetric extension of the Standard Model, *i.e.* the least number of new fields needed for the supersymmetric theory to work. In Table 1.2 we have listed the field content of the MSSM, all mass eigenstates except h^0 (which is the SM-like Higgs) are new supersymmetric particles.

Names	Spin	P_R	Gauge Eigenstates	Mass Eigenstates
Higgs bosons	0	+1	$H_u^0\ H_d^0\ H_u^+\ H_d^-$	$h^0\ H^0\ A^0\ H^\pm$
squarks	0	−1	$\tilde{u}_L\ \tilde{u}_R\ \tilde{d}_L\ \tilde{d}_R$	(same)
			$\tilde{s}_L\ \tilde{s}_R\ \tilde{c}_L\ \tilde{c}_R$	(same)
			$\tilde{t}_L\ \tilde{t}_R\ \tilde{b}_L\ \tilde{b}_R$	$\tilde{t}_1\ \tilde{t}_2\ \tilde{b}_1\ \tilde{b}_2$
sleptons	0	−1	$\tilde{e}_L\ \tilde{e}_R\ \tilde{\nu}_e$	(same)
			$\tilde{\mu}_L\ \tilde{\mu}_R\ \tilde{\nu}_\mu$	(same)
			$\tilde{\tau}_L\ \tilde{\tau}_R\ \tilde{\nu}_\tau$	$\tilde{\tau}_1\ \tilde{\tau}_2\ \tilde{\nu}_\tau$
neutralinos	1/2	−1	$\tilde{B}^0\ \tilde{W}^0\ \tilde{H}_d^0\ \tilde{H}_u^0$	$\tilde{\chi}_1^0\ \tilde{\chi}_2^0\ \tilde{\chi}_3^0\ \tilde{\chi}_4^0$
charginos	1/2	−1	$\tilde{W}^\pm\ \tilde{H}_u^+\ \tilde{H}_d^-$	$\tilde{\chi}_1^\pm\ \tilde{\chi}_2^\pm$
gluino	1/2	−1	$\tilde{g}$	(same)

Table 1.2: Particle content of the MSSM. Not all gauge eigenstates and mass eigenstates are the same. For the mass eigenstates that differ from the gauge eigenstates we have mixing between the gauge eigenstates giving mass eigenstates. Adapted from [14].

[14] A sneutrino also fulfills the same requirements but is excluded by direct detection searches [17]. A gravitino is also possible as a dark matter particle but is very weakly-coupling, and if R-parity is violated it is one of the few candidates.

We see in Table 1.2 that there are some particles with different gauge and mass eigenstates. This is due to mixing between gauge states with the same quantum numbers. However, even though the gauge and mass eigenstates are the same, it does not mean that the Lagrangian soft mass parameter (see Section 1.2.4) is the same as the physical mass. For the gluino, which is central to this thesis, the Lagrangian soft mass parameter is M_3, which is a running (scale dependent) mass. The physical (or scale-independent) mass at one-loop order is given by,

$$m_{\tilde{g}} = M_3(Q)\left(1 + \frac{\alpha_s}{4\pi}\left[15 + 6\ln\left(\frac{Q}{M_3}\right) + \sum A_{\tilde{q}}\right]\right), \tag{1.21}$$

where Q is the scale and the sum runs over all twelve squark-quark multiplets,[15] and the $A_{\tilde{q}}$ are given by

$$A_{\tilde{q}} = \int_0^1 \mathrm{d}x\, x \ln\left[x\frac{m_{\tilde{q}}^2}{M_3^2} + (1-x)\frac{m_q}{M_3^2} - x(1-x) - i\epsilon\right], \tag{1.22}$$

where $m_{\tilde{q}}$ is the squark mass and m_q the quark mass.

In general, due to supersymmetry breaking and electroweak symmetry breaking, the sfermion mass eigenstates are also a mixture of the gauge eigenstates. However, it can be shown that the mixing of sleptons and squarks is proportional to the masses, or Yukawa couplings, to be precise, of their SM partners. These couplings are very small for the first two generations, thus we can neglect the mixing for the first and second generation. For these two generations the masses are given on the form

$$m_F^2 = m_{F,\,\mathrm{soft}}^2 + \Delta_F, \tag{1.23}$$

where $m_{F,\,\mathrm{soft}}$ are contributions from $\mathcal{L}_{\mathrm{soft}}$ and Δ_F contributions due to electroweak symmetry breaking.

The third generation SM quarks and leptons have large masses and we therefore have significant mixing of the third generations sleptons, and squarks. Thus, the expressions for these sparticle masses are more complicated. We will not discuss the third generation squarks in this thesis since they are not present in initial states, only in loops, and they have thus a small effect on the production cross section.

1.2.4 The Soft Supersymmetric Lagrangian

As discussed in Section 1.2.1 we can divide the Lagrangian for a supersymmetric theory in two parts. In Eq. (1.24) we write down the soft terms in the general

[15] Six squarks-quark multiplets times two since they come in two versions: left and right handed.

MSSM Lagrangian.

$$\begin{aligned}
\mathcal{L}_{\text{soft}} = &-\frac{1}{2}\left[M_1\bar{\tilde{B}}^0\tilde{B}^0 + M_2\bar{\tilde{W}}_A\tilde{W}_A + M_3\bar{\tilde{g}}_B\tilde{g}_B\right] \\
&-\frac{i}{2}\left[M_1'\bar{\tilde{B}}^0\gamma_5\tilde{B}^0 + M_2'\bar{\tilde{W}}_A\gamma_5\tilde{W}_A + M_3'\bar{\tilde{g}}_B\gamma_5\tilde{g}_B\right] \\
&-\epsilon_{ab}\left[bH_u^aH_d^b + \text{h.c.}\right] - m_{H_u}^2|H_u|^2 - m_{H_d}^2|H_d|^2 \\
&+\sum_{i,j=1,3}\Big(-\Big[\tilde{Q}_i^\dagger(\mathbf{m}_Q^2)_{ij}\tilde{Q}_j + \tilde{d}_{\text{R}i}^\dagger(\mathbf{m}_d^2)_{ij}\tilde{d}_{\text{R}j} \\
&+\tilde{u}_{\text{R}i}^\dagger(\mathbf{m}_u^2)_{ij}\tilde{u}_{\text{R}j} + \tilde{L}_i^\dagger(\mathbf{m}_L^2)_{ij}\tilde{L}_j + \tilde{e}_{\text{R}i}^\dagger(\mathbf{m}_e^2)_{ij}\tilde{e}_{\text{R}j}\Big] \\
&-\epsilon_{ab}\Big[(\mathbf{T}_u)_{ij}\tilde{Q}_i^aH_u^b\tilde{u}_{\text{R}j}^\dagger - (\mathbf{T}_d)_{ij}\tilde{Q}_i^aH_d^b\tilde{d}_{\text{R}j}^\dagger \\
&-(\mathbf{T}_e)_{ij}\tilde{L}_i^aH_d^b\tilde{e}_{\text{R}j}^\dagger + \text{h.c.}\Big]\Big)\,.
\end{aligned} \tag{1.24}$$

Here i, j are generation indices, $A = 1, \ldots, 3$ and $B = 1, \ldots, 8$ are the gauge generator indices, and $a, b = 1, 2$ the $\text{SU}(2)_L$ indices.

Nearly all of the 105 parameters introduced from the general MSSM are contained in this soft supersymmetry breaking part of the Lagrangian, $\mathcal{L}_{\text{soft}}$, the only contained in the $\mathcal{L}_{\text{SUSY-SM}}$ is the μ-parameter. Thus it is the supersymmetry breaking that introduces a vast number of new parameters, not the supersymmetry itself. However, experimental constraints makes it possible to remove some of the parameters in the theory. Restrictions on flavour violating processes imply that we can set the Hermitian mass-squared matrices $\mathbf{m}_Q^2, \mathbf{m}_u^2, \mathbf{m}_d^2, \mathbf{m}_L^2$ and $\mathbf{m}_e^2$ in Eq. (1.24) to be diagonal to good approximation.

In the first and second line of Eq. (1.24) we have the gaugino masses, associated with the superpartners $\tilde{B}^0$, $\tilde{W}_A$, and $\tilde{g}_B$ of the SM gauge bosons. Due to experimental constraints on CP-violation we may as an approximation set $M_1' = M_2' = M_3' = 0$. The Higgs sector is described by the third line with the explicit mass terms $m_{H_u}^2$ and $m_{H_d}^2$ and the complex parameter b, as well as with the μ parameter in the supersymmetric part of the Lagrangian.

In the fourth and fifth line we have the explicit sfermion mass terms mentioned above. In the four last lines we have the trilinear scalar couplings between Higgs and squarks or sleptons. Flavour-changing and CP-violating processes are in experiments found to be very suppressed, thus we have to incorporate this into our theory. We assume $(\mathbf{T}_f)_{ij} = (\mathbf{A}_f)_{ij}(\mathbf{Y}_f)_{ij}$, with i, j being generation indices and $f = \{u, d, e\}$, where $(\mathbf{A}_f)_{ij}$ is a constant scalar, and $\mathbf{Y}_f$ are the Yukawa coupling matrices. We require that only the $(3, 3)$ components of each of the Yukawa matrices are relevant, *i.e.* we can approximate them as,

$$\mathbf{Y}_u \approx \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{1.25}$$

thus only $(\mathbf{T}_u)_{33}$ is important. In MSSM-24 we will then have A_t, A_b, and A_τ for $\mathbf{T}_u$, $\mathbf{T}_d$, and $\mathbf{T}_e$.

It is worth noting that there is no prescription here for how supersymmetry breaking occurs. We only introduce what seems to be an arbitrary part in our Lagrangian to explicitly break supersymmetry in order to have a phenomenological description of how the effective supersymmetry behaves.

We will work with a restricted MSSM-model named MSSM-24 which has 24 free parameters defined at a scale Q. We call this a weak-scale MSSM model since the scale Q is often set to something near the weak scale. From the discussion above we will arrive at the MSSM-24, let us count the parameters in Eq. (1.24) to prove that: We have three gaugino mass parameters from the two first lines, five parameters from the Higgs sector in the third line where two of them are restricted by EWSB, the diagonal sfermion mass matrices in fourth and fifth line gives 15 parameters, and finally we have three constant scalars in the two last lines giving in total 24 parameters. The parameters are listed in Table 1.3. The parameters in the Higgs sector are b, μ, $\tan\beta$, m_{H_d}, and m_{H_u}. Two parameters are restricted by EWSB due to the requirements of the minimum in the Higgs potential, while b is traded for m_A^{pole}. This gives us the parameters listed in Table 1.3.

Parameter	Property
M_1	$U(1)_Y$ gaugino mass
M_2	$SU(2)_L$ gaugino mass
M_3	$SU(3)_C$ gaugino mass
A_t	Top trilinear coupling
A_b	Bottom trilinear coupling
A_τ	Tau trilinear coupling
μ	μ parameter
m_A^{pole}	Pseudoscalar Higgs pole mass
$\tan\beta$	Ratio between electroweak VEVs
$\mathbf{m}_L^2$	Left handed slepton masses
$\mathbf{m}_e^2$	Right handed slepton masses
$\mathbf{m}_Q^2$	Left handed squark masses
$\mathbf{m}_u^2$	Up type right handed squark masses
$\mathbf{m}_d^2$	Down type right handed squark masses

Table 1.3: Parameters in the MSSM-24. A_f, where $f = \{t, b, \tau\}$, given $\mathbf{T}_f = \mathbf{A}_f \mathbf{Y}_f$. The five last parameters written in bold text are 3×3 diagonal matrices.

1.2.5 The Constrained Minimal Supersymmetric Standard Model

It is possible to constrain the many parameters of the MSSM in many ways. One much studied model is the *Constrained Minimal Supersymmetric Standard Model*.[16] This model has only five parameters,

$$m_0, m_{1/2}, \tan\beta, A_0, \text{sign}(\mu), \tag{1.26}$$

were m_0 is a common scalar mass, $m_{1/2}$ a common fermion[17] mass, $\tan\beta$ the ratio between the two Higgs vacuum expectation values H_d^0 and H_u^0, A_0 the common trilinear coupling constant, and μ the Higgsino mixing parameter.

It is believed that the spontaneous breaking of supersymmetry happens in a so-called *hidden sector* involving a gravitational mechanism. This happens at a

[16] Also called minimal supergravity, mSUGRA.
[17] Or gaugino if you want.

very high energy scale. The wish to unify all forces leads us to the constrained MSSM with the five parameters above. The parameters are defined at this very high energy scale, the corresponding sparticle masses at low energies are then found by using *renormalization group equations* which describe the running of the masses.

In any case the name *Constrained* Minimal Supersymmetric Standard Model is well motivated.

1.2.6 Higgs and Radiative Electroweak Symmetry Breaking

It is also worth talking about the Higgs fields in the MSSM. In the SM we had one physical Higgs field, while we in the MSSM have five physical fields. To recap briefly, the SM has a Higgs doublet with two complex scalar fields. These may also be desbribed as four real scalar fields. In electroweak symmetry breaking three of these real scalar fields are "eaten" by the $W^{\pm}$ and Z^0 such that they acquire masses, and we are left with one real scalar field giving the physical Higgs boson. In the MSSM it turns out that it is necessary to have two Higgs doublets in order to give up- and down-type quarks masses correctly.[18] Thus, we can now, analogously to the SM electroweak symmetry breaking, describe the two Higgs doublets as eight real scalar fields. In supersymmetry we still have the $W^{\pm}$ and Z^0 which acquire masses from three of the fields, and we are thus left with five physical Higgs bosons.

As mentioned, we need the electroweak symmetry breaking to be successful also in supersymmetry in order to give SM fermions and gauge bosons their masses. We want to break the $SU(2)_L \times U(1)_Y$ symmetry down to $U(1)_{\text{em}}$, to do this the scalar potential for MSSM in Eq. (1.27) must have three properties: i) It must be bounded from below, meaning that as we move far outward from the minimum the potential always increases, ii) it must have a minimum for non-zero field values, giving the potential the wine bottle shape,[19] iii) the minimum must have a remaining $U(1)_{\text{em}}$ symmetry.

$$\begin{aligned} V(H_u, H_d) =& |\mu|^2(|H_u^0|^2 + |H_u^+|^2 + |H_d^0|^2 + |H_d^-|^2) \\ & \frac{1}{8}(g^2 + g'^2)(|H_u^0|^2 + |H_u^+|^2 - |H_d^0|^2 - |H_d^-|^2)^2 \\ & \frac{1}{2}g^2|H_u^+ H_d^{0*} + H_d^0 H_d^{-*}|^2 \\ & m_{H_u}^2(|H_u^0|^2 + |H_u^+|^2) + m_{H_d}^2(|H_d^0|^2 + |H_d^-|^2) \\ & (b(H_u^+ H_d^- - H_u^0 H_d^0) + c.c.), \end{aligned} \tag{1.27}$$

where H_u^+, H_d^-, H_u^0, and H_d^0 are the complex scalar fields, g and g' are gauge couplings. μ, $m_{H_u}^2$, $m_{H_d}^2$, and b are the same as in Eq. (1.24).

The potential has eight degrees of freedom from the four complex scalar fields. To fulfill the three requirements above we can start by setting $H_u^+ = 0$ by using the gauge freedom in $SU(2)_L$. At the minimum we must thus have $\partial V/\partial H_u^+ = 0$ which leads to $H_d^- = 0$. We then have no vacuum expectation values for the charged fields, thus requirement iii) is fulfilled. To fulfill requirements

[18] "Correctly" means here to keep the superpotential holomorphic, which is a requirement necessary to keep the superpotential invariant under supersymmetry transformations, see Chapter 3.2 in [14].

[19] Cheers again!

i) and ii) the following inequalities must be obeyed,

$$2b < 2|\mu|^2 + m_{H_d}^2 + m_{H_u}^2, \tag{1.28}$$

$$b^2 > (|\mu|^2 + m_{H_d}^2)(|\mu|^2 + m_{H_u}^2). \tag{1.29}$$

Equation (1.28) ensures requirement i), while Eq. (1.29) fulfills ii).

1.2.7 Phenomenology

Up to now we have discussed some motivations for supersymmetry and what supersymmetry is. But how do we find it? Here we will discuss how the searches for supersymmetry at hadron colliders, such as the Large Hadron Collider (LHC), are performed and where today's limits on the sparticle masses are.

At the LHC protons (or, if you want, quarks and gluons) are being collided. Particles charged under QCD will thus have the largest cross sections, *i.e.* squarks and gluinos are what we should look for in the first instance. If R-parity is conserved squarks and gluinos will always be produced in pairs. Since they are unstable they will each decay into the LSP. The decay chain may be long and complicated but must provide colour charged particles, in other words we will see multiple jets. Other sources of jets are Standard Model processes involving quarks and gluons, the cross sections for these particles are much larger and will thus lead to large backgrounds. However, the LSP will escape the detector which we will see as missing energy in the transverse plane from an inbalance in momenta for the visible particles.[20] The way to look for the production of gluinos and squarks must therefore be to look for events with jets and large missing transverse energy.

When discussing searches for supersymmetric particles we often talk about exclusion limits. We also want to know what supersymmetry cannot be. In Fig. 1.2 we show exclusion limits for the CMSSM from an ATLAS search [18] at 8 TeV with 20.3 fb^{-1} of data. Here the expected limits have been calculated by setting the nominal event yield in each signal region to the corresponding mean expected background. In this search 15 signal regions were used. All signal regions requires missing transverse momentum above 160 GeV and that the effective mass is above 700 to 2200 GeV depending on the number of jets in the event. The number of jets allowed varies from two to six. There are several signal regions with the same number of jets, but they are distinguished with increasing background rejection. See Table 2 in [18] for an full overview.

We will later need the smallest interesting cross sections for a given energy and amount of data, *i.e.* the cross section values giving one event in a data set. To find this value we simply solve the equation

$$N = \sigma L, \tag{1.30}$$

where N is the number of events, σ the cross section, and L the integrated luminosity which is the amount of data available. Thus, for the existing 8 TeV data set with 20.3 fb^{-1} data we get with $N = 1$,

$$\sigma = \frac{1}{20.3 \text{ fb}^{-1}} \approx 0.05 \text{ fb}. \tag{1.31}$$

[20] When colliding hadrons we do not know the initial state momentum of the colliding gluons and quarks in the longitudinal direction, thus we cannot use energy conservation in the longitudinal direction.

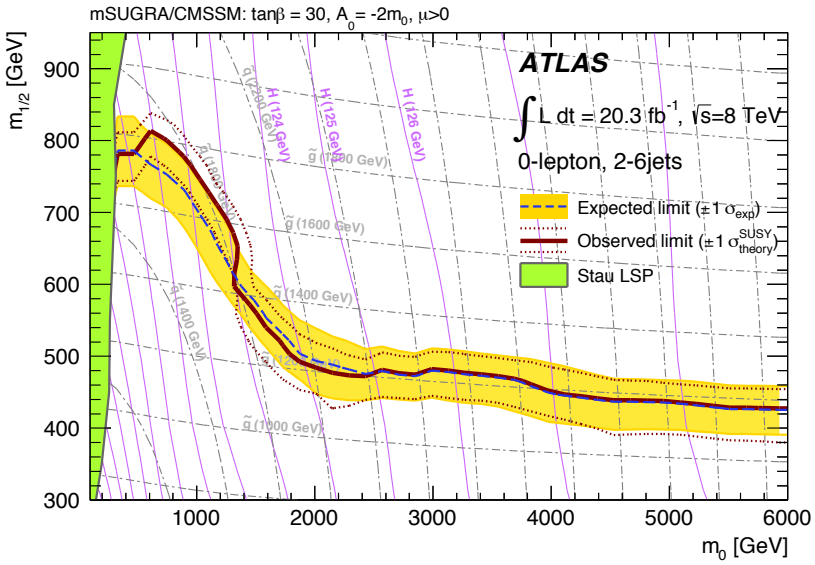


Figure 1.2: Exclusion limits for the CMSSM $(m_0, m_{1/2})$-mass plane in dark red, expected limits in dashed blue lines, assuming $\tan\beta = 30$, $A_0 = -2m_0$, and $\mu > 0$. Taken from [18].

$\sqrt{S}$	Amount of data	Cross section for $N = 1$
7 TeV	4.57 fb^{-1}	0.22 fb
8 TeV	20.3 fb^{-1}	0.05 fb
13 TeV	80 fb^{-1}	0.01 fb

Table 1.4: Cross section values corresponding to one event in different data sets used by the ATLAS collaboration at CERN. The amount of 13 TeV data is the most current value as the experiment is still running and continously collects data.

Relevant cross section values when discussing searches with a data set of 20.3 fb^{-1} are thus greater than or around 0.05 fb. In Table 1.4 we have listed the cross section values giving one event in different data sets. Note that even though the total integrated luminosity for 13 TeV data is around 80 fb^{-1} this does not mean that all of the data is used in each analysis done by the experiment, one often does a selection of the data relevant to the analysis.

Chapter 2

Calculation of the Gluino Pair Production Cross Section

We will in this chapter motivate the need for higher-order cross sections, review the analytical leading order cross section for gluino pair production, go through the numerical calculation of the next-to-leading order gluino production cross section in `Prospino 2.1` [1, 2], review the method `NLL-fast 2.1` [3, 4, 5, 6] uses and end the chapter discussing the achievable accuracy on the cross sections.

2.1 Why NLO Cross Sections and Why Fast?

First of all, why are next-to-leading cross sections necessary? From [2] we find good motivations for calculating σ_{NLO}: The leading order cross section of gluino pair production is heavily scale dependent, thus leading to uncertain predictions. By including the next-to-leading order cross section we will reduce the scale dependence and thus improve the precision of the theoretical predictions.

To quantify the correction of the leading order cross section we may use the K-factor, $K = \sigma_{\text{NLO}}/\sigma_{\text{LO}}$. In Fig. 2.1 the K-factor is plotted as a function of gluino mass. The squark mass is defined by the given ratio, and we have four cases where the squark mass varies from being twice the gluino mass down to 80% of the gluino mass. It is assumed here that the squark mass is degenerate. We see that NLO diagrams have large contributions at all masses, thus σ_{NLO} should be used instead of σ_{LO} to get a realistic estimate of the total cross section.

One reason to do as precise calculations as possible is that we want to find as strong exclusion limits as we can, *i.e.* what supersymmetry cannot be. An example of such an exclusion limit is shown in Fig. 2.2, which can be compared to Fig. 1.2, where the program `ColliderBit` [19] has been used to calculate LHC observables for the CMSSM. It shows exclusion limits on m_0 and $m_{1/2}$ (briefly discussed in Section 1.2.5) from ATLAS data where only LO cross sections are included in the `ColliderBit` scan (white line) and where NLO+NLL cross sections are included in ATLAS' results (blue line). ATLAS used the `NLL-fast 2.1` tool, which we discuss in Section 2.4. We see clearly the difference in exclusion limits, data with NLO+NLL cross sections excludes a larger part of the parameter space than LO cross section data.

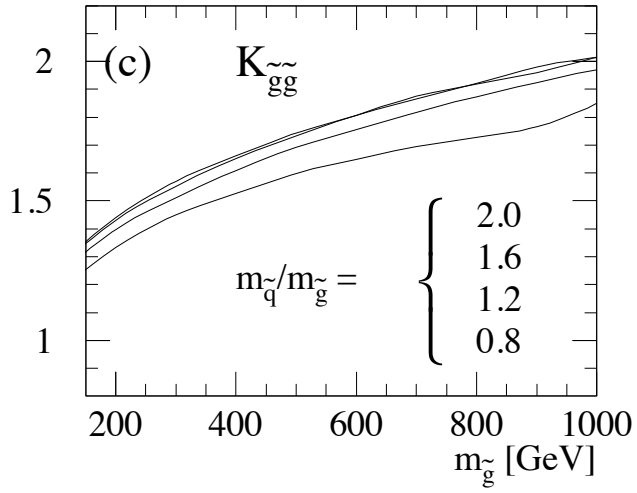


Figure 2.1: K-factor plotted as function of the gluino mass for different ratios between the gluino and squark mass with center-of-mass energy at 14 TeV. The K-factor increases with the gluino mass. Here the gluino mass is maximally 1 TeV, while we will study gluino masses of up to 4 TeV. From [2]

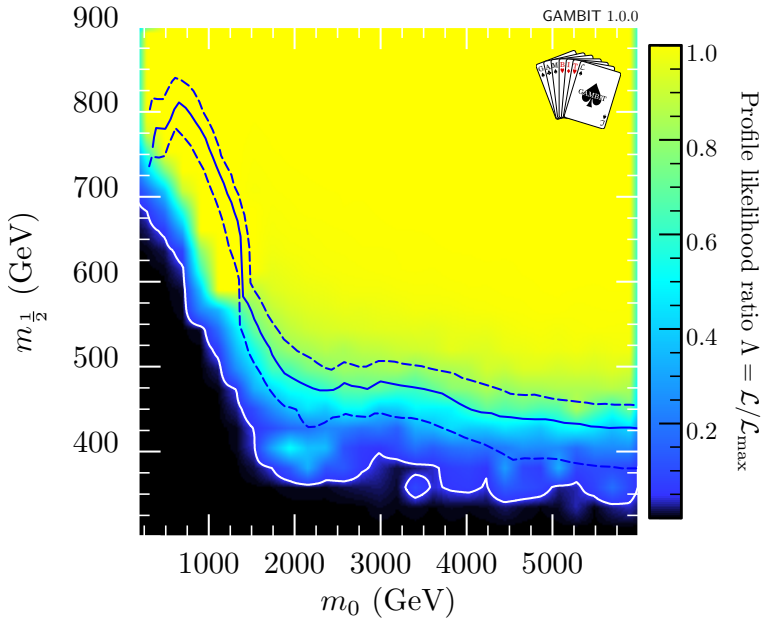


Figure 2.2: Exclusion limits on m_0 and $m_{1/2}$ in the CMSSM, with $\tan\beta = 30$, $A_0 = -2m_0$, and $\mu > 0$, calculated with leading order terms in `ColliderBit`, plotted as a white line together with exclusion lines from ATLAS-data in blue using NLO+NLL. Including next-to-leading order corrections would have increased the precision on the results from `ColliderBit`. Figure taken from [19].

Using `Prospino 2.1` one can calculate next-to-leading order cross sections. However, it takes some time. Each cross section evaluation for all strong pro-

duction processes in the CMSSM takes ~ 15 minutes. When doing a scan, for example the one in Fig. 2.2, it is necessary to calculate at least 10^4 cross sections, multiplying that with 15 minuts gives a very large number with units of time. Therefore a *fast* evaluation of cross sections at the highest possible accuracy is desirable.

2.2 Gluino Pair Production Cross Section

So how do we perform the calculation with today's methods? The partonic leading order cross section at Born-level is possible to calculate analytically and it is,

$$\sigma^B(gg \to \tilde{g}\tilde{g}) = \frac{\pi\alpha_s^2}{s}\left[\beta_{\tilde{g}}\left(-3 - \frac{51m_{\tilde{g}}^2}{4s}\right) - \left(\frac{9}{4} + \frac{9m_{\tilde{g}}^2}{s} - \frac{9m_{\tilde{g}}^4}{s^2}\right)\log\left(\frac{1-\beta_{\tilde{g}}}{1+\beta_{\tilde{g}}}\right)\right] \tag{2.1}$$

$$\begin{aligned}\sigma^B(q\bar{q} \to \tilde{g}\tilde{g}) = {} & \frac{\pi\alpha_s^2}{s}\beta_{\tilde{g}}\left(\frac{8}{9} + \frac{16m_{\tilde{g}}^2}{9s}\right) \\ & + \frac{\pi\alpha_s\hat{\alpha}_s}{s}\left[\beta_{\tilde{g}}\left(-\frac{4}{3} - \frac{8m_-^2}{3s}\right) + \left(\frac{8m_{\tilde{g}}^3}{3s} + \frac{8m_-^4}{3s^2}\right)L_2\right] \\ & + \frac{\pi\hat{\alpha}_s^2}{s}\frac{32}{27}\left[\beta_{\tilde{g}}\left(1 + \frac{m_-^4}{m_{\tilde{q}}^2 s + m_-^4}\right) - \left(\frac{2m_-^2}{s} + \frac{m_{\tilde{g}}^2}{4(s-2m_-^2)}\right)L_2\right],\end{aligned} \tag{2.2}$$

where $m_{\tilde{g}}$ is the gluino mass, $m_{\tilde{q}}$ the squark mass which is assumed to be degenerate, $\sqrt{s}$ the center of mass energy, and,

$$L_2 = \log\left(\frac{s - 2m_-^2 - s\beta_{\tilde{g}}}{s - 2m_-^2 + s\beta_{\tilde{g}}}\right), \tag{2.3}$$

$$\beta_{\tilde{g}} = \sqrt{1 - \frac{4m_{\tilde{g}}^2}{s}}, \tag{2.4}$$

$$m_-^2 = m_{\tilde{g}}^2 - m_{\tilde{q}}^2, \tag{2.5}$$

$$\alpha_s = \frac{g_s^2}{4\pi}, \qquad \hat{\alpha}_s = \frac{\hat{g}_s^2}{4\pi}, \tag{2.6}$$

where g_s and $\hat{g}_s$ is the qqg and $q\tilde{q}\tilde{g}$ couplings respectively. These couplings are equal. As we will discuss in Chapter 4 we will use the partonic leading order cross section to help the machine learning algorithm learn the next-to-leading order gluino pair production cross section. From Eqs. (2.1) and (2.2) we see that using $\beta_{\tilde{g}}$, m_-, and L_2 as inout can help the machine learning algorithm since they provide much of the structure of the LO partonic cross section.

The more cumbersome NLO cross section requires loop integrals with poles and therefore numerical tools are used. Currently, cross section results are calculated with the state-of-the-art `Prospino 2.1` program [1, 2], which we use as the basis for our data generation discussed in Section 4.1.

Since protons are collided at LHC we need the gluino pair production cross section from an initial state of two protons. When colliding protons we have

to take into account that the proton is a composite particle, it consists of three quarks *uud* (two up and one down quark), which we call *valence quarks*, and a sea of *virtual* quarks and gluons. These virtual quarks and gluons arise from QCD interactions. Each constituent of a hadronic particle, like the proton, is called a *parton*. Thus, when colliding protons each parton will have a fraction of the proton's total momentum. To describe the distribution of the fraction of the total momentum for the partons in a proton we use the *parton distribution functions* (PDF). These functions are not known analytically since QCD is non-perturbative at the scale of the proton binding energy and must therefore be determined experimentally. The main behaviour of a PDF is that for low energy scales the valence quarks in a proton will be the most dominant, while for large energy scales the sea quarks and gluons will play a larger role in the interaction taking place.

In practice, when we calculate hadronic cross sections with protons as initial state we must therefore first calculate the partonic cross section, $\sigma(i,j \to \tilde{g}\tilde{g}; s)$ with i,j being the initial state partons, and $\sqrt{s}$ the partonic center-of-mass energy, and then integrate it over the possible fractions x_1, x_2 of total proton momenta for each initial state parton,

$$\sigma_{\text{TOT}} = \iint \mathrm{d}x_1 \mathrm{d}x_2 \, f_i(x_1, Q^2) f_j(x_2, Q^2) \sigma(i,j \to \tilde{g}\tilde{g}; s = x_1 x_2 S), \tag{2.7}$$

where $f_i(x_1), f_j(x_2)$ are the PDFs, Q the renormalization scale, and $\sqrt{S}$ is the total center-of-mass energy.

There are many possible choices of PDFs, one can do fits to different data sets and different perturbative order, in this thesis we use the standard NLO PDF CTEQ6.6M [20] when running `Prospino 2.1` and `NLL-fast 2.1`.

2.3 Outline of Calculations in Prospino 2.1

The total hadronic cross section is found by calculating the integral of the double differential cross section over rapidity y and transverse momentum p_t,

$$\sigma(pp \to \tilde{g}\tilde{g} + X) = \int_{p_t^{\min}}^{p_t^{\max}} \mathrm{d}p_t \int_{y^{\min}}^{y^{\max}} \mathrm{d}y \, \frac{\mathrm{d}^2\sigma(\tilde{g}\tilde{g})}{\mathrm{d}p_t \mathrm{d}y}. \tag{2.8}$$

In case of other processes, such as a final state $\tilde{q}\tilde{g}$, it's worth noting that y and p_t is defined as the rapidity and transverse momentum of the second particle. The double differential cross section can be written as,

$$\frac{\mathrm{d}^2\sigma}{\mathrm{d}p_t \mathrm{d}y} = 2p_t S \sum_{i,j=g,q,\bar{q}} \int_{x_1^-}^1 \mathrm{d}x_1 \int_{x_2^-}^1 \mathrm{d}x_2 \, x_1 f_i^{h_1}(x_1, Q^2) x_2 f_j^{h_2}(x_2, Q^2) \frac{\mathrm{d}^2\hat{\sigma}_{ij}(x_1 x_2 S, Q^2)}{\mathrm{d}t \mathrm{d}u}. \tag{2.9}$$

where t and u are the Mandelstam variables. `Prospino 2.1` calulates Eq. (2.9) at both LO and NLO. The NLO result includes the sum of leading and next-to-leading order contributions.

2.3.1 VEGAS Integration Routine

`Prospino 2.1` uses the VEGAS integration routine by Lepage [21] for calculating the cross section integrals. This routine is a Monte Carlo integration routine

where one combines *importance sampling* and *stratified sampling*. Importance sampling is a sampling method that use a probability density function that resembles the integrand such that one samples more often from areas where the contribution to the integral is large. Stratified sampling is a method where we divide up the integration interval into subintervals and sample with an equal number of points in each subinterval, the trick here is to choose the subintervals carefully such that the total variance is reduced.

Both of these sampling methods requires a knowledge of the integrand, which we do not necessarily have in all cases. Lepage introduced the VEGAS routine that combines these two sampling methods into a routine that do several sets of evaulations of the integrand and adjusts the probability density function after each set of evaluations. By doing it this way it samples points where the contribution to the integral is large without knowing the function form *a priori*. Other methods have been proposed as well, but these turns out to perform poorly in dimensions ≥ 4 as demonstrated in [21].

We will follow [22] *ad verbum*. Consider an integral,

$$I = \int_\Omega \mathrm{d}^n x \, f(\vec{x}). \tag{2.10}$$

This integral is in standard Monte Carlo integration estimated by,

$$I \sim S = \frac{1}{N} \sum_{i=1}^{N} \frac{f(\vec{x}_i)}{p(\vec{x}_i)}, \tag{2.11}$$

where $p(\vec{x})$ is the probability density which the random points are drawn with. The main idea behind VEGAS is to do M estimates S_m of I using N evaluations of the integrand. The estimates are combined to give a cumulative estimate $\bar{S}$,

$$I \sim \bar{S} = \bar{\sigma}^2 \sum_{m=1}^{M} \frac{S_m}{\sigma_m^2}, \tag{2.12}$$

where σ_m is the approximate uncertainty of S_m and $\bar{\sigma}$ is the approximate uncertainty of $\bar{S}$,

$$\sigma_m^2 = \frac{1}{N-1} \left(\frac{1}{N} \sum_{i=1}^{N} \frac{f(\vec{x}_i)^2}{p(\vec{x}_i)} - S_m^2 \right), \tag{2.13}$$

$$\frac{1}{\bar{\sigma}^2} = \sum_{m=1}^{M} \frac{1}{\sigma_m^2}. \tag{2.14}$$

Theoretically σ_m^2 is minimized when

$$p(\vec{x})_{\text{optimal}} = \frac{|f(\vec{x})|}{\int_\Omega \mathrm{d}^n x |f(\vec{x})|}, \tag{2.15}$$

which means that sample points are concentrated where the contribution to the integral is largest.[1] The approach in VEGAS for minimizing σ_m^2 is to divide the integration volume into hypercubes forming a rectangular grid. Random points

[1] The denominator in Eq. (2.15) is really a normalization factor here.

are then chosen from the different hypercubes with equal probability. From iteration to iteration the delimiters between the hypercubes are adjusted such that $p(\vec{x}) \to p(\vec{x})_{\text{optimal}}$.

In addition to the estimate of the integral, the algorithm also tells us whether the estimates are consistent by computing the χ^2 per degree of freedom (dof),

$$\frac{\chi^2}{\text{dof}} = \frac{1}{M-1} \sum_{m=1}^{M} \frac{(S_m - \bar{S})^2}{\sigma_m^2}. \tag{2.16}$$

If the algorithm gives good results $\frac{\chi^2}{\text{dof}}$ should not be much larger than one. If not the estimates does not agree within errors.

The relative numerical error on the integrals performed in `Prospino 2.1` is found to be typically of the order $10^{-2} - 10^{-3}$ in our data set. The VEGAS integration routine is thus a reliable and accurate integration routine also when using it to calculate the next-to-leading order cross sections in supersymmetry.

2.4 NLL-fast 2.1

`NLL-fast 2.1` [3, 4, 5, 6] is a computer program which computes cross sections for squark and gluino production up to so-called NLO+NLL precision.

The *next-to-leading-log* (NLL) corrections come from the soft gluon resummation described in [4]. The threshold region for producing (anti)squarks or gluinos is reached when the center of mass energy $\sqrt{S} \to 4m^2$, where m is the average mass of the produced particles. In this region, where the final state particles move slowly we have large corrections to the leading order cross section from Coulomb corrections, *i.e.* gluon exchanges between slowly moving particles, and soft gluon corrections due to emission of low energy gluons from the colored initial and final states. We can describe the soft gluon corrections by powers of large logarithms of the final state particles' velocity β, thus we get the names leading- and next-to-leading-log.

To calculate a cross section `NLL-fast` reads in two-dimensional grid files with squark and gluino masses, their corresponding NLO, NLO+NLL cross sections, scale uncertainty, PDF, and α_s error, then it performs a interpolation between the grid points to find the desired cross section. The interpolation routine used is a polynomial interpolation subroutine from Numerical Recipes [23].

A cross section is calculated in a few seconds, which is good compared to `Prospino 2.1` which may use several minutes, however, `NLL-fast 2.1` is restricted to degenerate squark masses. The usefulness of `NLL-fast 2.1` is therefore limited, in particular for less constrained models with many free parameters.

2.5 Accuracy of today's methods

From the discussion in [19] we can see that the error due to higher order corrections has been as low as 10% when including NLO+NLL corrections. Simplifying the matter a bit, we may say that this error is an estimate which is calculated by varying the renormalization and factorization scale, the so-called scale ucertainty. The more the cross section varies when doing this the larger the error is.

But we must also take into account the errors from PDFs and α_s. The PDFs have larger errors at high energies and large momentum fractions x which is problematic for heavy sparticles. The reason for poor PDFs at high momentum fractions is that there is less data to construct the PDFs from for those energies. However, as data is collected at the LHC the PDFs will improve in this region. An example with `NLL-fast 2.1` (8TeV) gives us errors of $(24.3\%, -22.2\%)$ for the PDF and $(8.3\%, -7.3\%)$ for α_s with gluino and squark mass at 1.5 TeV for gluino pair production. To improve this result a better combined fit of PDFs and α_s to LHC data is needed.

From this we can conclude that the method developed in thesis must have systematic errors below 10% in order to give a result which is better than the current methods.

Chapter 3

Machine Learning

In this chapter we will introduce the concept of machine learning (ML). We will look at the basics of ML, then briefly discuss neural networks before introducing the gradient boostied decision tree algorithm. We will then discuss overtraining and other challenges with ML, and how we decide whether a ML model is performing well or not.

3.1 Basic Concepts

We have two categories of ML: so-called *supervised* and *unsupervised* learning. Supervised learning is when we have a dataset with *features* and *targets*, *i.e.* we know the answer (target) for a parameter point (features) and the algorithm can learn a map from features to targets. Unsupervised learning is a technique for discovering hidden structures in a dataset. An example is to use unsupervised learning on images, if you for example have many pictures of dogs, the algorithm will try to find what is similar between the pictures and eventually manage to predict when a picture contains a dog.

Two subcategories of both supervised and unsupervised learning are *classification* and *regression*. Classification ML is classification of samples into a restricted set of categories, *e.g.* decide whether or not there is a dog in a picture or a Higgs boson in an LHC collision. Regression ML is when the output values can take continuous values, put in other words, we predict values of a function from a set of known values of that function. In this thesis we will focus on supervised learning with regression.

A possibly problematic issue with machine learning is *overtraining*. If we do supervised learning and run the training algorithm too long the model will eventually learn the training data rather than the underlying relationships in the training data. It is important to check this while one builds a ML prediction model. This is in contrast to the problem with *underfitting* which is when we have a model which does not describe the underlying relationships in the data. When doing training on a machine learning model we have to divide the data set into two parts, one for training and one for validation of the final model. This is often called *training set* and *validation set*[1] If one perform *k-fold cross validation* (CV) the machine learning model is trained several times by using

[1] Also called hold-out set.

a fraction of the training set, the training set is divided into k folds where one fold is reserved for testing and the other $k-1$ folds are used for training. The scores[2] on the precision from the k models are then averaged and outputted. With this procedure it is possible to lower the bias from how you choose your training and data set. At the very end we can perform the final validation with the above-mentioned validation set.

The *hyper parameters* are the parameters defining the ML model. The training procedure consists of adjusting the hyper parameters and how to use the training data optimally. Adjusting the hyper parameters can be automatized by using a scanning method to test out a set of hyper parameters. The training data should also be cleaned before training the predictive model. Outliers in the data set should be identified and the data set must be cover the parameter space sufficiently, if there are gaps or sparsely populated regions the learning may be incomplete. If we have a data set with many variables we should consider which of them to use in the training, irrelevant variables will slow down and give inefficient training.

3.2 Artificial Neural Network

A popular example of a ML algorithm is the *artificial neural network* (ANN, or neural network for short) first described by McCulloc and Pitt [24]. It is inspired by the biologial neural network formed by neurons in the human brain. An ANN consists of artificial neurons which can take a value nominally between 1 and 0, has a weight and an activation function, for example the sigmoid function,

$$\sigma(v) = \frac{1}{1+e^{-v}}. \tag{3.1}$$

The neurons are organized in layers and are connected with each other from layer to layer, an illustration is shown in Fig. 3.1. We have one or more *input neurons* which are connected to the next layer which is a *hidden layer*, there can be several of these, while the last layer is the *output layer* which gives the output of the network.

For the case of regression there can be several neurons that can output any value, in case of classification there is one output neuron for each class. The number of input neurons depends on the number of variables fed into the network. The neurons in a neural network can be connected in different ways, *e.g.* fully or sparsely connected. We have a fully connected neural network when a neuron is connected with all neurons in the previous and next layer. A sparsely connected network have fewer connections between the neurons. The weights of the neurons changes as the learning proceeds. Thus, after training the network all the neurons have gained different weights and the neurons will trigger depending on the input.

3.3 Gradient Boosted Decision Trees

The other example we will look at, and the algorithm we will use in this project, is the boosted decision tree (BDT). A decision tree is a predictive model which

[2]There are many ways to calculate the precision of ML models, but they have in common that they measure the precision in a single number we call the score.

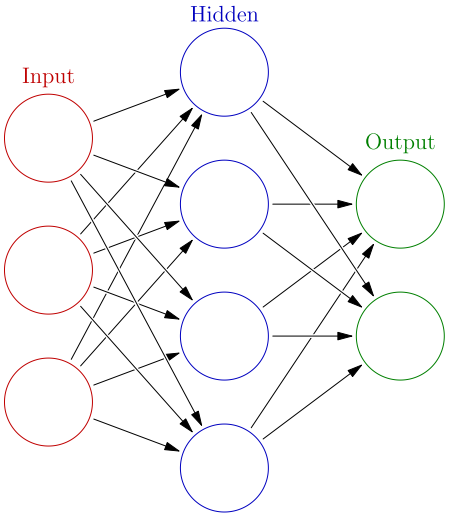


Figure 3.1: Illustration of a neural network. Taken from [25].

can be used on large datasets. It is based on the simple classification method where we draw a tree with different criteria and walk through the graph to make the correct decision. We will here focus on regression trees. When discussing decision trees we can illustrate this behaviour by a square which represents the feature space (X_1, X_2) of a function $f(X_1, X_2)$ with output Y. See Fig. 3.2. We can split this space into J regions, *e.g.* at $X_1 = t_1$ and $X_2 = t_2$ and so on such that we get the situation at the top right of Fig. 3.2. We name the *terminal regions*[3] as $R_1, \ldots, R_J$. Note that we simplify matters by always choosing a binary split which is easier to describe than the situation at the top left of Fig. 3.2.

The response, or output, of the regression tree $f(X_1, X_2)$ inside region R_j is modelled as a constant γ_j, which we can write as

$$f(X_1, X_2) = \sum_{j=1}^{J} \gamma_j I\{(X_1, X_2) \in R_j\}, \tag{3.2}$$

where we use the indicator notation $I\{x \in R\}$ which evaluates to 1 if the statement inside the brackets is true and 0 otherwise.

Fitting a tree to data consists of finding the terminal regions R_j by splitting the training data into different parts. As mentioned above binary splits are used. The training data is split with the use of a splitting variable j and a split point s which give us the half planes,

$$R_1(j, s) = \{X | X_j \leq s\} \quad \text{and} \quad R_2(j, s) = \{X | X_j > s\}. \tag{3.3}$$

Now we have to choose a minimization criterion in order to find the best splits. If we use the minimization of the sum of squares, $\sum(y_i - \gamma_i)^2$ with y_i being the

[3] Also often called *terminal nodes* or *leaf nodes*.

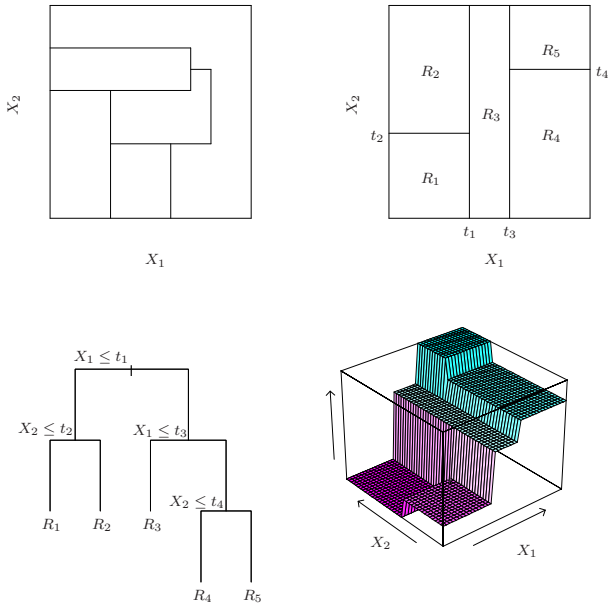


Figure 3.2: Illustrations of a decision tree. Top right shows a tree obtained by binary splitting, while the top left figure is not obtained that way. The former is easier to describe mathematically and thus also to implement in a program. At the bottom left the binary split decision tree is drawn as a tree, and at the bottom right we see a 3D illustration of the prediction surface of the tree. Figure taken from [26].

target values, we want the splitting varible j and split point s to minimize,

$$\min_{j,s}\left[\min_{\gamma_1}\sum_{X_i\in R_1(j,s)}(y_i-\gamma_1)^2+\min_{\gamma_2}\sum_{X_i\in R_2(j,s)}(y_i-\gamma_2)^2\right]. \tag{3.4}$$

The minimization criterion is also called the *loss function*, which we will discuss in Section 3.6, the criterion used here is the *least squares loss*. The minimization of the sums in Eq. (3.4) is thus for any choice of j and s solved by,

$$\gamma_1=\text{mean}(y_i|X_i\in R_1(j,s)),\qquad \gamma_2=\text{mean}(y_i|X_i\in R_2(j,s)), \tag{3.5}$$

i.e. each terminal region output the mean of the target values of the samples inside the terminal region. This procedure with splitting the data set is then repeated until we have reached the maximal number of terminal regions we have set.

More generally, when considering trees with many input variables $X=(X_1,\ldots,X_N)$, we can write the tree as,

$$T(X_i;\Theta)=\sum_{j=1}^{J}\gamma_j I(X_i\in R_j), \tag{3.6}$$

where $\Theta=\{R_j,\gamma_j\}_1^J$ describe the terminal regions and their assigned constants. We want the Θ which gives us the smallest loss between predictions and target

values,

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \sum_{j=1}^{J} \sum_{X_i \in R_j} L(y_i, \gamma_j), \tag{3.7}$$

with L being the loss function, and argmin means that we pick the Θ that minimize the sum. Eq. (3.7) can be seen as a generalized version of Eq. (3.4) describing all terminal regions R_j and constants γ_j. If R_j is fixed it is easy to fit γ_j, but finding the optimal R_j is usually a very computationally expensive task. As we will discuss below, one approach is to change the criterion such that one approximates R_j by minimizing some other loss criterion instead of the one used in Eq. (3.7).

The decision tree we have discussed up to now consists of only one tree. A method to improve this decision tree model is to use a *boosted decision tree*. Boosting is simply the procedure of adding trees that corrects the previously added trees in the model. We can write a BDT as $f_M(X_i) = \sum_{m=1}^{M} T(X_i; \Theta_m)$, where each decision tree $f_m(X_i) = T(X_i; \Theta_m)$ has its own set of $j = 1, \ldots, J$ terminal regions R_{jm} and corresponding constants γ_{jm}. The criterion for finding the best Θ_m at the m-th iteration is,

$$\Theta_m = \underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(X_i) + T(X_i; \Theta_m)), \tag{3.8}$$

where we sum over all training points y_i, $f_{m-1}(X_i)$ is the current model and $T(X_i; \Theta_m)$ the new tree added. Equation (3.8) is the main equation to solve at each training iteration m.

In the gradient boosting algorithm shown in Alg. 1 the approach is to initialize a tree model that predicts the mean[4] of the targets and to improve it by adding new trees. After the first tree is initialized the algorithm will fit M trees. Each fit is done by using *gradient descent*[5] which calculates the gradient r_{im} of the loss function for each training point with target y_i. The gradient is found by differentiating the loss function L with respect to the current model. In the case of least squares loss we will get,

$$\nabla L = 2(y_i - f_{m-1}(X_i)) = r_{im}. \tag{3.9}$$

By fitting a tree to these gradients we step in the direction in loss function space that minimize the loss function most. This step is the simplification of Eq. (3.8), fitting a tree to the *gradients* of the loss function instead of the *actual* loss function turns out to be a good approximation of the terminal nodes. A more detailed discussion is found in [26].

When the terminal regions are found we can fit the corresponding γ_{jm}. This is done by minimizing the loss between the targets and current model in each terminal region R_{jm}. The last step is to update the model by adding the new tree with terminal regions R_{jm} and constants γ_{jm}.

[4] If we use least squares loss. When using least absolute deviance and Huber loss it predicts the α-quantile of the targets. See Sec. 3.6 for an introduction to least absolute deviance and Huber loss functions.

[5] Also named *steepest descent*

1. Initialize $f_0(x) = \text{argmin}_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$;
2. **for** $m = 1$ *to* M **do**
   a) **for** $i = 1, 2, \ldots, N$ **do**
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$
   end
   b) Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \ldots, J_m$;
   c) **for** $j = 1, 2, \ldots, J_m$ **do**
   $$\gamma_{jm} = \underset{\gamma}{\text{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$
   end
   d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
end
3. Output $\hat{f}(x) = f_M(x)$;

Algorithm 1: Algorithm for the gradient boosting. We are using the implementation in `Scikit-Learn`[27].

3.4 Hyper Parameters

Now that we have discussed the gradient boosting decision tree algorithm, we can take a look at the hyper parameters defining the model. The most important ones are the number of *boosting iterations* (how many trees to add), the *depth* (longest path from root to terminal region) of the trees, and the *learning rate* (adjusting the contribution of each tree added). The learning rate $0 < \nu \leq 1$ is simply a scale factor multiplied to the new tree when adding it to the model,

$$f_m(X_i) = f_{m-1}(X_i) + \nu \cdot \sum_{j=1}^{J_m} \gamma_{jm} I(X_i \in R_{jm}), \tag{3.10}$$

where $f_{m-1}(X_i)$ is the existing model, and $\sum_{j=1}^{J_m} \gamma_{jm} I(X_i \in R_{jm})$ the new tree added to the model.

The *subsample* parameter decides whether we use all of the training samples in each iteration or a subsample of them. By using a subsample of the tranining sample we reduce the bias and correlation between each tree. To further adjust the model we can adjust the sensitivity for when we make a split in the tree, how many branches and leaves we will have in total, and how many of the input features the tree should take into account at the same time in each iteration. In Table 4.3 we have listed the hyper parameters used in this thesis.

There are few guiding principles on how to choose the parameters. However, we can look at some of the implications of the different parameters. Obviously, the number of boosting iterations will impact the run time of the training. If we have a high learning rate we can have fewer iterations, but the tree may not be sensitive to small features in the data set. To compensate we can adjust the max depth of the tree, if we have a deep tree we get more sensitivity to small

variations in the data set, however, we may then become more vulnerable for overtraining, see below. If we have complicated relations between input and output it may be good with a large depth. With these considerations in mind we can start the training, but there is no way to tell *a priori* which parameters that will turn out to be optimal.

3.5 Quantification of Performance

When we build a BDT we must quantify the performance in a reasonable way. `Scikit-Learn` offers several ways to do this. In this section we will discuss the different measures, how and where to use them, and define our own measure: the mean relative deviance.

3.5.1 Precision Measures

When building several BDTs we would like to quantify the performance in a single number in order to be able to easily compare the different versions. There are several ways to do this, for example using the median absolute error, which is robust to outliers,[6] or the score method we will use in this thesis: the R^2-value, which gives a measure on how well future samples will be predicted by the BDT. The R^2-value is defined as,

$$R^2 = 1 - \frac{u}{v}, \tag{3.11}$$

where $u = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ and $v = \sum_{i=1}^{N}(y_i - \bar{y})^2$, in which $\hat{y}_i$ is the predicted value, y_i is the target value and $\bar{y}_i$ is the mean of the target values. u is called the *residual sum*, *i.e.* the sum of the deviations between model and data. v is called the *total sum of squares* and is the squared difference between true data and the overall mean, it describes the variation in target values. The R^2 score provides a measure of how well future samples are likely to be predicted correctly. $R^2 \leq 1$ and saturates when all samples are predicted correctly.

After picking out the best model based on the R^2-value, we must check for overtraining. This is done by looking at the deviance between the training data and target values, and the deviance between the test data and target values as functions of the number of iterations. The deviance is defined as the value of the loss function. The sweet spot for the model is the point where both the training set and test set deviance are at their lowest. As we will see later, overtraining has not been a large problem for us. We have instead seen that the deviance graphs for the training data set and test data set have diverged at a large number of training iterations, an example is shown in Fig. 3.3. We call this *inefficient training*, it does not decrease the precision of the predictions. It is, however, better when the deviance graphs track each other, then each training sample is more efficently used.

Once we have checked whether the model is overtrained or not, we can look more closely at how the model performs. In our case we want to generate a model that predicts cross sections for an LHC-process as described in Chapter 2. Typically, cross sections vary over many orders of magnitude, thus it is crucial

[6] Data points that are far from being predicted correctly.

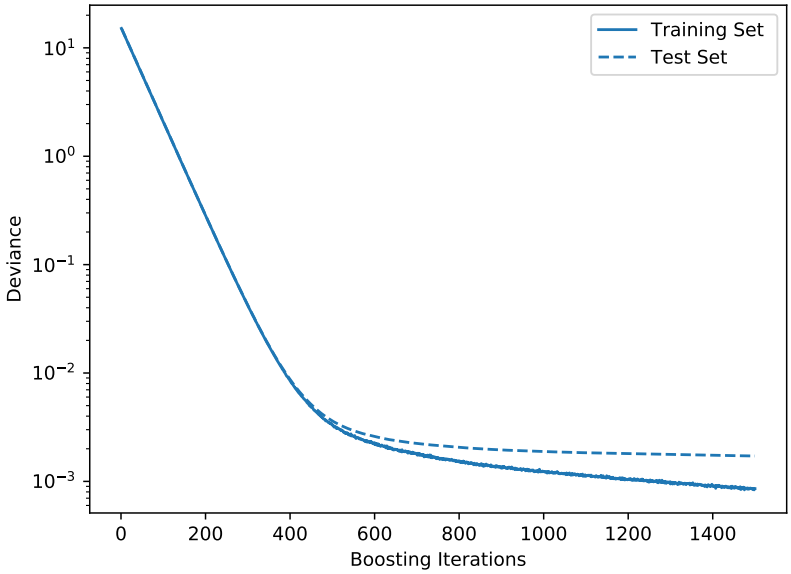


Figure 3.3: Example of a deviance plot. The dashed line is the deviance between predictions on the test data and true values, while the solid line is the deviance between predictions from the training data and true values. In this case we do not have overtraining since the test set slope is still negative. However, the deviance curves diverge. This is a sign of inefficient training.

to test how well our models predicts the cross section at each decade.[7] The problem with the built-in measures in `Scikit-Learn` is that they are measures on the model as a whole. This problem is solved by introducing the *mean relative deviance*,

$$\bar{\epsilon} = \frac{1}{N}\sum_{i=1}^{N}\frac{\hat{y}_i - y_i}{y_i}, \tag{3.12}$$

where $\hat{y}_i$ is the predicted value and y_i is the target value. The mean relative deviance is calculated for each decade. We also calculate the standard deviation σ_ϵ of the relative deviance,

$$\sigma_\epsilon = \sqrt{\frac{1}{N}\sum_{i=1}^{N}|\epsilon_i - \bar{\epsilon}|^2}, \tag{3.13}$$

where ϵ_i is the relative deviance for a data point y_i. Note that the mean relative deviance (and its standard deviation) is calculated with the *actual cross section values*, not $\log_{10}(\sigma)$ that the model is trained on, for each decade of the cross section values. We will discuss this more in Section 4.2.

[7] A decade is in this thesis defined as the half-open interval $(10^n, 10^{n+1}]$. We will often work with the base-10 logarithm of the cross sections, thus we will use the notation $(n, n+1]$ for this decade.

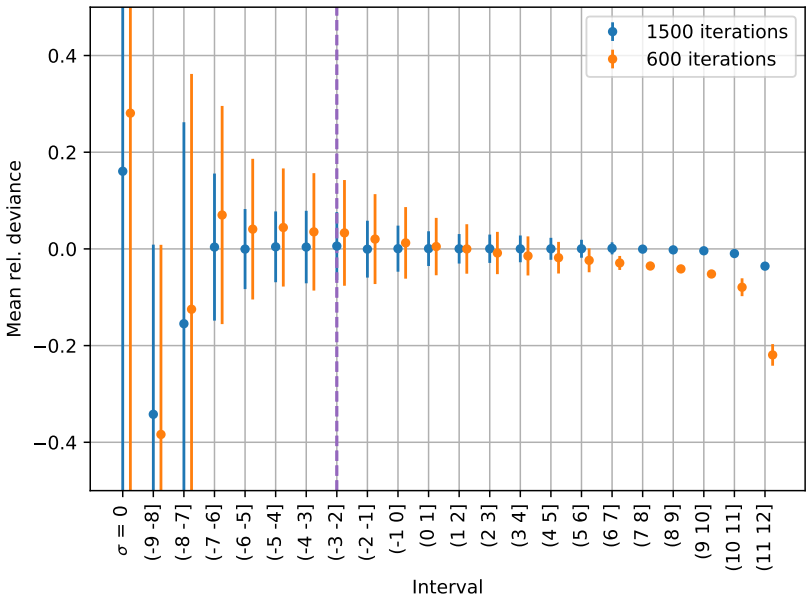


Figure 3.4: Example showing how the mean relative deviance for each decade changes when the number of boosting iterations is varied. As expected we see that the BDT model predicts better when it is trained longer, the standard deviation of the relative deviance is lower when we have a large number of iterations. Note also that the slope of the points is changing. With fewer iterations the BDT predicts systematically above and below the true values, depending on whether the cross sections are small or large. When the number of iterations is increased the slope decreases.

If we have $\bar{\epsilon} \neq 0$ it means that the BDT predicts systematically below or above the true cross section. We could in principle correct for this by rescaling the results in each decade, but that is not desirable. We will plot the mean relative deviance per decade with standard deviation as error bars, an example is shown in Fig. 3.4. The purple dashed line represents 0.02 events in a 20.1 fb^{-1} data set.[8] We see how the mean relative deviance develops when changing the number of boosting iterations. With few iterations we may get a model that has an overall $\bar{\epsilon} = 0$, but when we look at the values for each decade of cross section values we see that high cross sections are underestimated, while small cross sections are overestimated.

In addition to $\bar{\epsilon} = 0$ our goal is to have $\sigma_\epsilon < 0.1$. If σ_ϵ is below 0.1 it means that the spread in relative deviance is mostly inside 10%. In [19] the errors in the currently available calculations of the cross sections are discussed, they are of the order 10% or larger, which means that if we have a BDT model where the error is significantly below 10%, the errors from other sources such as the the PDFs and α_s will dominate the errors introduced by the BDT-model.

It is worth noting that even though we could *test* our model with more

[8] Apologies for the arbitrary number of events.

samples we will not get any better value for σ_ϵ. What will happen is that we get a more precise measure of $\bar{\epsilon}$ and σ_ϵ, the only way to improve the BDT's precision is by tuning the hyper parameter and/or adding more training data.

3.5.2 Variable Importance

Since we are working with physics we expect and want a BDT-model that behaves in accordance with the physics it describes. We are here interested in the gluino pair production cross section. It is therefore natural to think that the gluino mass will play an important role, thus we expect a BDT model where the gluino mass parameter has a large impact. We will also expect that the squarks contribute in a certain order depending on their flavour. Since the first generation quarks are more important in the PDF than the second generation squarks, we expect that the first generation squarks will be the most important. In Fig. 3.5 we see an example of a *variable importance* plot. In this case the variables are contributing as expected from a physics point of view.

The variable importance is calculated as a relative measure on which training variables that affects the model most. At each split in the trees in the model only one input variable is considered at the time, thus the more splits a training variable causes, the more important it is. The measure is normalized.

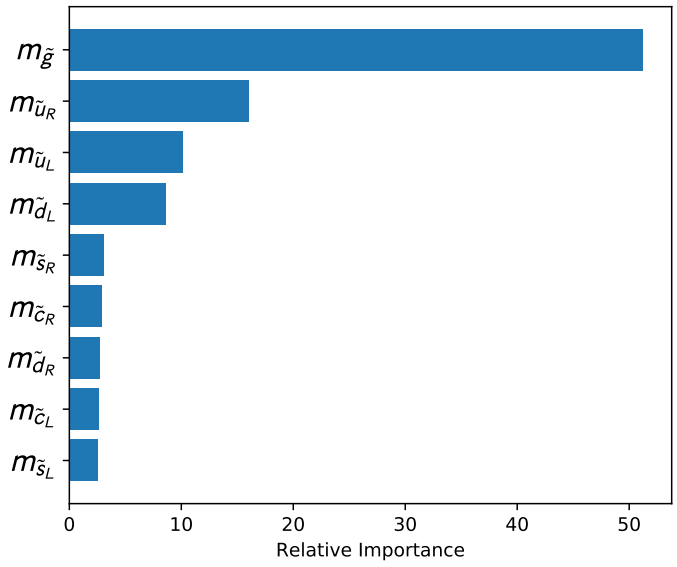


Figure 3.5: Variable importance for an example BDT model which is trained with physical gluino and first and second generation squark masses. We see here a reasonable result where the gluino mass is the most important variable, while the squarks contribute less. The lightest squarks contribute the most, while the second generations squarks are the lowest contributing variables.

3.5.3 Learning Curves

In addition to the deviance plots we can also plot *learning curves*. These curves tells us how well we have utilized the training data. An example is shown in Fig. 3.6. The *cross validation score* shows how well our model performs on the test data set, while *training score* is how well our model performs on the training data set. The scores are the R^2 value defined in Eq. (3.11).

Each point on the curve represents a model trained with the amount of training data indicated on the x-axis. A cross validation method splits the total data set in three folds, as discussed in Section 3.1. For each iteration a subset of the total data is used, the first iteration uses 1/5 and the last iteration uses all of the data. Thus each point in the plot represents three models[9] trained with a subset of the data increasing in size for each iteration, *i.e.* the first models is trained with 1/5 of the data, the second with 2/5 of the data, and so on up to the last models where the whole data set is used. Thus the leftmost red point in Fig. 3.6 is the mean of the training scores of the three models trained with a fifth of the total data set, the leftmost green point is then the mean of the cross validation scores from the three models. The envelope around the graphs is the standard deviation of the scores.

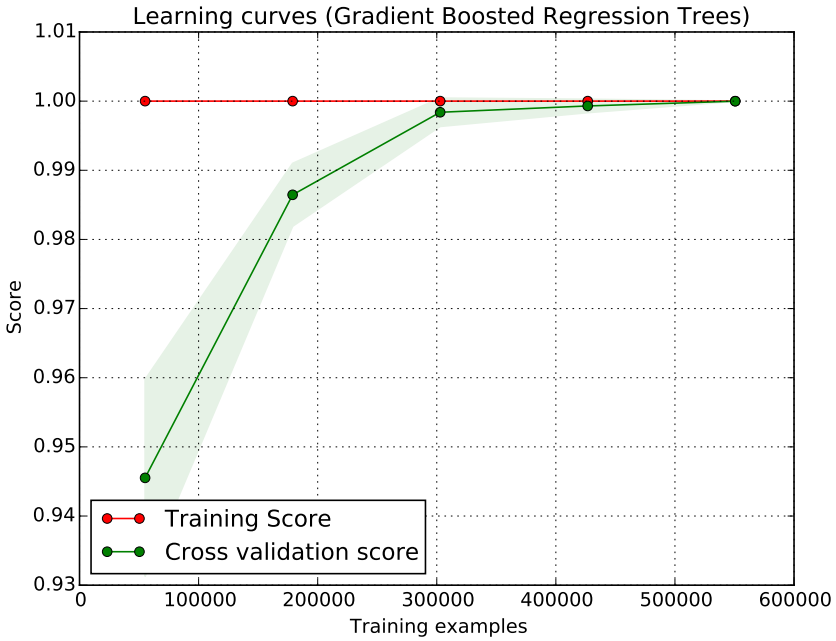


Figure 3.6: Learning curve with R^2 scores against data points plotted with built in functionality in `Scikit-Learn`. An example BDT is shown where we have utilized the training data well. In this case the hyper parameters of the model are well tuned and we benefit from adding more data up to around 500 000 points.

[9]Remember that with k-fold cross validation we train k models such that each fold is used as a test set once.

This plot is used to investigate whether the model benefits from more data or not. If the two graphs converge to a small score the precision of the model can be increased by changing the hyper parameters of the model such that it becomes more sensitive to features in the data. If the training score is high and the cross validation does not fully climb to the training score and if it not flattens out, the model would benefit from adding more data to the training.

3.6 Loss Functions

When building a BDT we have to choose a loss function to use when training the model. The choice of loss function will impact the time needed for training and the quality of the model. In gradient boosting we add new trees to the model that minimize the loss function, in other words: trees that decrease the difference between the predicted and true values.

There are four loss functions available in `GradientBoostingRegressor`, we will take a look at three of them. The default loss function is least squares loss (LS). The other choices are least absolute deviation (LAD), Huber [28] which is a combination of the first two, and quantile regression. For situations with few outliers, the least squares loss is likely to perform well and give us a good BDT model with little training since it has a quadratic penalty for deviations between prediction $\hat{y}$ and target y,

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2. \tag{3.14}$$

If we have outliers, we can use the least absolute deviance function. It gives us a linear loss, which is more robust for the outliers since the outliers will not dominate the loss as much as with least squares,

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|. \tag{3.15}$$

If we combine these two such that the outliers are covered by the linear loss and the other samples by the squared loss, we arrive at the Huber loss function,

$$L(y, \hat{y}) = \begin{cases} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2, & |y_i - \hat{y}_i| \leq \gamma \\ \sum_{i=1}^{N} (\gamma(|y_i - \hat{y}_i| - \frac{\gamma}{2})), & \text{otherwise} \end{cases} \tag{3.16}$$

The γ in (3.16) sets the demarcation between squared and linear loss. We define how large fraction α of the predictions should have least squares loss, then we find $\gamma = \alpha$-quantile$\{|y_i - \hat{y}_i|\}$. Thus, if $\alpha = 0.9$ it means that the 90% best of all the predictions should have least squares, while the remaining 10% worst of all the predictions should have linear loss. `Scikit-Learn` uses a method from `Scipy` [29] to find γ in each case. This γ is only calculated the first time we call the `LossFunction`-object, thus data points will move inside the region $|y_i - \hat{y}_i| \leq \gamma$, where LS loss is used, as training progresses.

When using these loss functions we want to compare the performance of each of them. However, it is not straight forward to do so, since we must take into account the fact that the squared loss is squared and that the linear loss is

linear. We will square the value of the linear loss in order to compare the two loss functions. For the Huber loss, which is a mix of squared and linear loss, we cannot do this easily. There is a factor of two that differs in the definition the Huber loss in the `Scikit-Learn` implementation from the two other loss functions, so we must multiply it by two before comparison. At the end of the training we assume that the set of predictions with $|y_i - \hat{y}_i| > \gamma$ will be very small, thus we can assume we have a pure LS loss at the end.

Chapter 4

Evaluating Supersymmetric Cross Sections With Boosted Decision Trees

In this chapter we will present how we use BDTs to evaluate supersymmetric cross sections at next-to-leading order. First we will look at the software used to generate data samples and the quality controls of the data. We will then delve into the mysterious world of tuning a BDT model by using grid searches. The choice of input variables is then discussed before we conclude the chapter with the final manual tuning of our BDTs.

4.1 Data Generation

In this section we will first discuss the details of the data generation for the MSSM-24 training data, then briefly go through the generation of CMSSM data used to test our BDT model. After that we will present the main steps to build a BDT model. All scripts to build and plot results from the BDT models are available on GitHub: `https://github.com/jonvegards/master_thesis`

Sampling of data To generate the data we used a `MPI` parallelized `Python` script [30] modified to suit this thesis. The script generates a sample point in the parameter space for a given supersymmetric model by drawing random parameter values from given intervals, see Table 4.1 for parameters and intervals used. Then the sample is passed over to `softpoint.x`, a script in the `Softsusy 3.6.2`-package [31], which calculates the supersymmetric spectrum. A parameter point where we get successful electroweak symmetry breaking (see Section 1.2.6), a neutralino as the LSP, and the absence of tachyons is accepted.

The input and output from `softsusy.x` is written in the SLHA-format [32], which `Prospino 2.1` takes as input. `Prospino 2.1` calculates the LO and NLO cross sections for the specified supersymmetric process at a given center-of-mass energy with the specified initial state.[1] The LO and NLO cross sections, the K-factor, and absolute and relative error on the NLO cross section from `Prospino`

[1] Initial state also decides type of collider. We may use $p\bar{p}$ (Tevatron) or pp (LHC.)

`2.1` is written into the SLHA-file for each parameter point. When we have generated "enough" data samples, we harvest the interesting values[2] from the SLHA files into a single data file which we will use to train the BDT model with.

We note that `Prospino 2.1` first calculates the LO and NLO cross sections assuming degenerate masses in order to find an estimate for the K-factor, *i.e.* calculating $K = \sigma_{\text{NLO, degenerate}}/\sigma_{\text{LO, degenerate}}$. Then it proceeds with calculating the LO cross section assuming non-degenerate squark masses, thus the output NLO cross section is estimated with $\sigma_{\text{NLO}} = \sigma_{\text{LO}}K$.

Prior When sampling data by drawing random values from parameter intervals we have to think carefully how to ensure that the parameter space is well covered. We will train a BDT to predict NLO cross sections for gluino pair production, thus we only have to care about the *cross section function and its parameter space*, *i.e.* we want to cover the parameter space of the cross section function well. We will use two methods of sampling in this thesis, the discussion of these in [33] is more thorough.

In a Bayesian approach, when considering parameters for a model we must encode what we know, or do not know, about the parameter in the *prior probability distribution*, which is where we express our belief in a parameter value. From the principle of *transformation group invariance*, which means that the prior should remain unchanged under any transformation irrelevant for the problem, we can extract two types of prior distributions. The first one is found by considering a *location parameter* x. For such parameters under a constant coordinate transformation $x \to x' = x+a$ the prior $\pi(x)$ should remain unchanged,

$$\pi(x)\,\mathrm{d}x = \pi(x')\,\mathrm{d}x' = \pi(x+a)\,\mathrm{d}(x+a) \Rightarrow \pi(x) = \pi(x+a), \tag{4.1}$$

since $\mathrm{d}x = \mathrm{d}(x+a)$. This is satisfied if $\pi(x)$ is a *uniform prior*, *i.e.* the probability of getting x or $x+a$ is the same.

The type of other prior is due to *scale parameters*. Such parameters introduce a scale to our problem. These parameters should have a prior that is invariant to scalings: $x \to x' = mx$. The requirement for the prior distribution will then be,

$$\pi(x)\,\mathrm{d}x = \pi(x')\,\mathrm{d}x' = \pi(mx)\,\mathrm{d}(mx) \Rightarrow \pi(x)\,\mathrm{d}x = \pi(mx)m\,\mathrm{d}x, \tag{4.2}$$

which is satisfied if $\pi(x)$ is a *log prior*, that is, a prior which is proportional to $1/x$. The name comes from that $\pi(\log(x))$ behaves as a flat prior.

A flat prior will cover the edges of the parameter space,[3] while the log prior cover the inner part of the parameter space. However, we cannot use the log prior as is. We use cuts to avoid divergence of the log prior near zero. An illustration is shown in Fig. 4.1. In the sampling script this is implemented with start and stop values for the sampling interval with additional cuts where we go from log prior to flat prior near zero. In Table 4.1 we have listed parameters and their corresponding sampling intervals for flat and log priors. The log prior

[2] Interesting in this context means variables related to the LO and NLO cross sections for gluino pair production.

[3] Consider the following: the probability for having a sample with n parameter values in the lower half of a uniform sample interval scales as $(1/2)^n$, where n is the number of parameters. This will be a very small probability the more parameter values we are sampling.

intervals are of the kind [start,flat_start,flat_stop,stop], where the two (or one, depending on whether we sample negative values or not) middle values are the point(s) where we go from log to flat prior.

As discussed in Section 1.2.4, we work with a weak scale MSSM model, more precisely the MSSM-24, thus we must set a scale Q for the soft supersymmetric breaking Lagrangian parameters. We use $Q = 1$ TeV. It is also worth noting that we sample data points in order to calculate the gluino pair production cross section at next-to-leading order, *i.e.* estimating the NLO cross section function. Thus we need only values for parameters in the cross section function. The leading order partonic cross section functions in Eqs. (2.1-2.2) contains five independent variables, $m_{\tilde{g}}$, $\tilde{g}_s$, $\hat{g}_s$, s, $m_{\tilde{q}}$. Thus we have twelve variables when considering only first and second generation squarks.[4] This means that we sample in a 24-dimensional space in search for values for a twelve-dimensional function. Thus we cover the function space better than it would appear from sampling a 24-dimensional space.

Parameter	Log prior range	Flat prior range
M_1	$[0, 100, 4000]$	$[0, 4000]$
M_2	$[0, 100, 4000]$	$[0, 4000]$
M_3	$[0, 100, 4000]$	$[0, 4000]$
A_t	$[-4000, -100, 100, 4000]$	$[-4000, 4000]$
A_b	$[-4000, -100, 100, 4000]$	$[-4000, 4000]$
A_τ	$[-4000, -100, 100, 4000]$	$[-4000, 4000]$
μ	$[-4000, -100, 100, 4000]$	$[-4000, 4000]$
m_A^{pole}	$[0, 100, 4000]$	$[0, 4000]$
$\tan\beta$	$[2, 60]$	$[2, 60]$
m_{L_1}	$[0, 100, 4000]$	$[0, 4000]$
m_{L_2}	$[0, 100, 4000]$	$[0, 4000]$
m_{L_3}	$[0, 100, 4000]$	$[0, 4000]$
m_{e_1}	$[0, 100, 4000]$	$[0, 4000]$
m_{e_2}	$[0, 100, 4000]$	$[0, 4000]$
m_{e_3}	$[0, 100, 4000]$	$[0, 4000]$
m_{Q_1}	$[0, 100, 4000]$	$[0, 4000]$
m_{Q_2}	$[0, 100, 4000]$	$[0, 4000]$
m_{Q_3}	$[0, 100, 4000]$	$[0, 4000]$
m_{u_1}	$[0, 100, 4000]$	$[0, 4000]$
m_{u_2}	$[0, 100, 4000]$	$[0, 4000]$
m_{u_3}	$[0, 100, 4000]$	$[0, 4000]$
m_{d_1}	$[0, 100, 4000]$	$[0, 4000]$
m_{d_2}	$[0, 100, 4000]$	$[0, 4000]$
m_{d_3}	$[0, 100, 4000]$	$[0, 4000]$

Table 4.1: Table showing the sampling intervals used for the parameters when sampling the MSSM-24 model, the scale is set to $Q = 1$ TeV. The log priors have three (four) limit values, the first and last are always start and end points for the sampling interval, while the one (two) in the middle is the limit where we go from flat prior to log prior. All values in GeV except $\tan\beta$ which is unitless.

[4] Four variables from the cross section expression and eight squark masses.

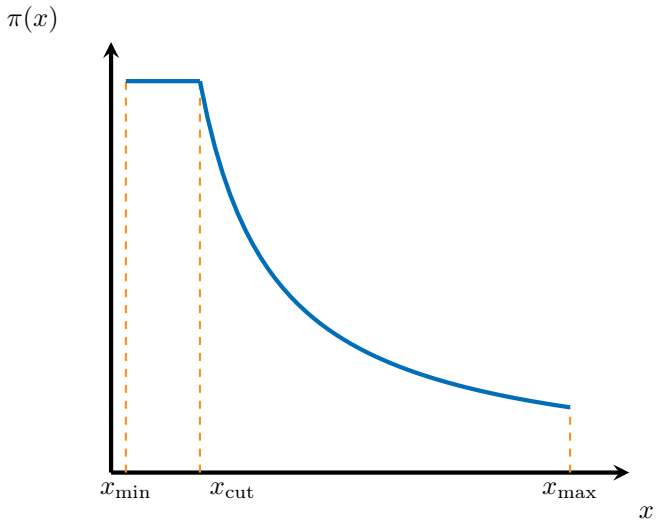


Figure 4.1: Illustration of a log prior distribution from x_{min} to x_{max}. Near zero the log prior blows up and we change to a flat prior at x_{cut}.

Challenges during data generation When we generated data we faced several challenges. One of them was to cover the parameter space thoroughly, this we solved by sampling with both flat and log priors as discussed above. Other challenges was to sample enough data without using an unreasonable amount of time, and to do quality checks of the data sets.

To save time we ran a parallelized sampling script. We wanted to generate 500 000 data samples each with log and flat prior, thus parallelization was necessary given a runtime of ~ 60 seconds per parameter point. However, when using more than $\sim$ 160 CPUs we had so many file operations that the data generation process slowed down . It turned out that the script `softpoint.x` was the source of the excessive file operations, mostly due to repeated library loading before calculating the spectrum for each proposed point. We reduced some of the file operation overhead by instead using the ELF[5] `lt-softpoint.x`. `softpoint.x` is a script that loads libraries *etc.* to make sure the execution of the main program runs without problems. The number of file operations are somewhat lower with `lt-softpoint.x`, but there are still many operations. Since we wanted to generate a total of 10^6 data samples we had to do it in several runs in order not to overload the computer cluster with file operations.

A subtle error in the data sets arose from `Prospino 2.1`. In `Fortran` codes one must hardcode the max length of a variable's value in file output. For floats in non-exponential notation this means that the maximal values is set the maximal number of digits. If this limit is exceeded we will get garbage output. This limitation lead to a limit on the K-factor in `Prospino 2.1`, it could not print out K-factors ≥ 10. Because of this many samples were discarded which created gaps in our data set. It is worth noting that a $K > 10$ implies a very large correction to the leading order cross section. Due to processes not present

[5] An Executable and Linkable Format (ELF) file is a standard file format for executable files. [34]

at leading order, it is in some cases possible with such large corrections.

Quality checks of data We have plotted the data as scatter plots in terms of input parameters and sparticle masses in order to see whether we had covered the desired mass ranges and to detect outliers arising from computational problems. In Fig. 4.2 we see some of the plots used, a scatter plot of the physical squark masses $m_{\tilde{u}_L}$ vs. $m_{\tilde{d}_R}$, and $m_{\tilde{c}_L}$ vs. $m_{\tilde{g}}$. Eeach color in the plot to the right represents a decade of the cross section values, the rightmost blue-grey band represents samples with zero cross sections, then the cross sections increase as we move leftwards. A full set of quality check plots is available at the GitHub-page for this project.

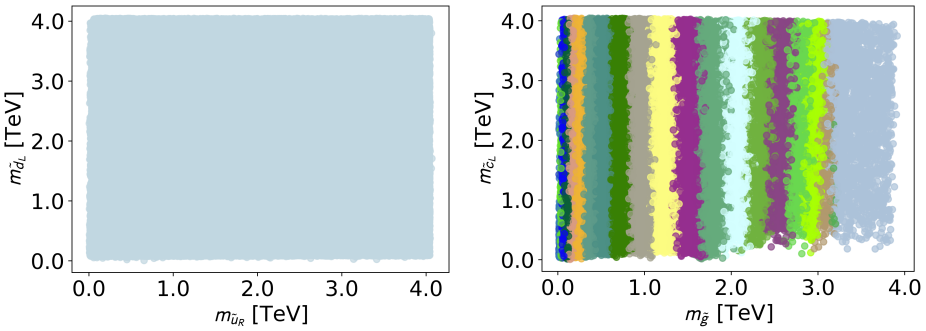


Figure 4.2: Sample quality control plots of the MSSM-24 data. To the right each color represents a decade of cross section values, with the blue-grey color to the right representing the smallest cross section values.

Generating Test Data We also generated 10^4 samples with CMSSM data reserved for testing the final model and comparing it to `NLL-fast 2.1`, see Section 5.5. The procedure described above was applied also for this data set, however, we only used a flat prior here. We also had the problem arising from large K-factor here, it was visible as a sharp gap in the scatter plots of the parameters. It disappeared when we implemented the solution discussed above. In Table 4.2 we have listed the parameters in CMSSM and their corresponding sampling intervals.

Parameter	Flat prior range
m_0	$[10, 4000]$
$m_{1/2}$	$[10, 4000]$
A_0	$[-4000, 4000]$
$\tan\beta$	$[2, 60]$
$\text{sign}(\mu)$	—

Table 4.2: Parameters sampled for the CMSSM data. All values except $\text{sign}(\mu)$ and $\tan\beta$ in GeV. The sign of μ had equal probability for being positive or negative.

4.2 Building a BDT

When building a BDT there are four main steps: choosing software, pre-processing data, choosing the input variables to use, and tuning the hyper parameters defining the BDT. We will discuss tuning of the hyper parameters in the next section and then the choice of input variables in Section 4.4. We will start here with discussing the software used and pre-processing of the data.

To do the data reading and manipulating we used `Pandas` [29]. This package has several features which makes the job easy for us. Once we have a comma (or other separator) separated file we can use `Pandas` to read in and treat data efficiently. In this project each of the two datafiles contains $\sim 5 \cdot 10^5$ samples.[6] When reading the data we split it in two parts, one with features and one with targets, as discussed in Chapter 3. We use 90% of the total data as training data, and save the remaining 10% for validation, *i.e.* the data we use when plotting the mean relative deviance. The features are the input values we give to the model, *e.g.* squark and gluino masses, while the targets are the desired output value(s), in our case it is the NLO gluino pair production cross section.

For building the BDT model we chose `Scikit-Learn` [27]. `Scikit-Learn` is an open source package for machine learning in `Python`, with a large number of machine learning algorithms built in. We use the `GradientBoostingRegressor` in this project.

When building a BDT model the choice of hyper parameters and input variables is essential. We have to do smart choices depending on our problem. Since we want to predict cross sections from supersymmetric data samples we may assume that the underlying relationships in the data are complex, thus using hyper parameter values giving many boosting iterations and deep trees are likely to give us a better model. In general a complex model will be able to learn complex relationships.

But we should also consider how we can pre-process the data in such a way that we simplify the training task for the algorithm. By considering the span of cross section values and the disscusion in Section 3.5, we chose to use $\log_{10}(\sigma)$ as the target.[7] This means that we must choose what to do with samples where $\sigma = 0$ fb before taking the logarithm. It is not necessarily a good idea to set samples with $\sigma = 0$ fb to for example $\sigma = 10^{-50}$ fb (which is close to zero), since we then will have a cluster of outliers in the data set. A better solution is to set $\sigma = 0$ samples to a cross section value close to, but smaller than the lowest cross section value in the data set. We chose to set $\sigma = 0$ samples equal to the power of 10 nearest to and less than the smallest cross section in the data set.

When the data is pre-processed we can start building our BDT model. The hard part is to find a suitable set of hyper parameter values. In Table 4.3 we have listed the hyper parameters relevant to us. After choosing a set of hyper parameter values we initialize the model and then fit it to our data.

The three main lines in a BDT model script using `GradientBoostingRegressor` are:

```
params = {'learning_rate': 0.01, 'loss': 'lad', 'max_depth': 13,
```

[6] One file where the MSSM-24 parameters are sampled with flat prior, and one where they are sampled with log prior.

[7] When taking the logarithm of a cross section we implicitly divide the cross section by a reference value $\sigma_0 = 1$ fb.

```
          'n_estimators': 1500}
reg = sklearn.ensemble.GradientBoostingRegressor(**params)
reg.fit(features, target)
```

In the first line we define the set of hyper parameter values. The hyper parameters chosen here are the ones we believe have the largest impact on the training, as we will discuss in the next section. After the training procedure is finished we can do predictions and test the model. There are several ways to test the model, either by built-in methods or self-made methods as discussed in Section 3.5.

Hyper parameter	Description
`n_estimators`	Number of boosting stages to perform.
`loss`	Loss function to be optimized.
`learning_rate`	The contribution of each tree.
`max_depth`	Maximum depth of individual regression estimator, limits the total number of nodes in the tree.
`subsample`	Fraction of samples to be used at each boosting stage.
`max_leaf_nodes`	Maximal number of terminal nodes in each tree.
`verbose`	If equal 1 more information is printed out while the training proceeds.
`random_state`	Seed used by the random sampling. Ensures that we can reproduce results exactly.

Table 4.3: Hyper parameters we used in the `GradientBoostingRegressor`. In Section 3.4 we discussed which hyper parameters that had most influence on the result.

4.3 Searching for the Optimal Hyper Parameters

In the search for an optimal BDT model manual tuning is a rather slow and inefficient method since the hyper parameter space can be arbitrarily large. Therefore we made use of the search method `GridSearchCV` in `Scikit-Learn` to help us navigate in the hyper parameter space. We will now discuss the process of tuning the hyper parameters, starting with the early trial and error before moving on with the grid search method.

When choosing which hyper parameters to train with we have to do some try and fail, there are few guiding principles on what to choose. However, as discussed in Chapter 3 the three hyper parameters `n_estimators`, `learning_rate`, and `max_depth` have large impact and we started by exploring them. Our first try on hyper parameter values was[8]

```
params = {'n_estimators': 300, 'max_depth': 5, 'random_state': 42,
          'learning_rate': 0.05, 'loss': 'ls', 'verbose': 1}.
```

Early trial and error with varying the hyper parameter values defined above showed that a high `n_estimators` and low `learning_rate` was a good combination. Other parameter did not influence the model as much as those mentioned,

[8]The hyper parameters `random_state` and `verbose` are only there for practical purposes. The former ensures that we can exactly reproduce the model each time and the latter gives us more information about the training procedure.

for example the hyper parameter `subsample`. However, as long as it was smaller than 1 the BDT did slightly better, we therefore fixed it to be 0.5. We used this information to form our grid of hyper parameter values to use in the grid search.

`GridSearchCV` takes as input an estimator, a dataset, and a parameter grid. The output is the best scoring estimator which is found by training BDTs with all possible combination of hyper parameters and comparing them. The estimator is the algorithm we want to use, here the `GradientBoostingRegressor`. The dataset consists of a $N \times M$ `NumPy` array where each row element corresponds to a point in the N-dimensional parameter space of the supersymmetry model we are looking at. The parameter grid is a `Python` dictionary with the hyper parameters and their corresponding values we want to try out in the search. The inputs for `GridSearchCV` are listed in Table 4.3.

Input	Description
`estimator`	Which estimator to use (`GradientBoostingRegressor`)
`param_grid`	Grid of parameters given as a dictionary with lists of values
`scoring`	Which scoring method to use. `GradientBoostingRegressor` uses the R^2 value by default.
`n_jobs`	Number of jobs for parallellization
`cv`	Which cross validation generator to use, default: k-fold CV
`verbose`	How much information do we want while the grid search is running

Table 4.4: Hyper parameters used in the `GridSearchCV`-function.

As we can see in Table 4.4 it is also possible to choose a score method to compare all the models. As discussed in Section 3.5 we use the R^2-value which is the default score in `GradientBoostingRegressor`. It is important to note that one may arrive at a model with poorer R^2-value when using hyper parameter values found by `GridSearchCV` than using manually tuned hyper parameters. This is due to `GridSearchCV` using cross validation, see Section 3.1. The cross validation procedure ensures that the BDT does not overfit to the training data by shuffling the data set such that the model does not train and test with the same portion of the data each time. Thus when we tune the parameters manually we may get a slightly higher R^2-value unless we also imitate the cross validation procedure.

Our approach was to first perform coarse searches and then successive searches with a finer hyper parameter grid. By doing it this way we could get a sense of where in hyper parameter space the best BDT models could be built. The loss functions were explored separately, see Section 4.5. We used the knowledge from our early trial and error to form our first grid of hyper parameter values to explore:

```
param_grid = {'n_estimators': [100, 300, 500, 700],
              'learning_rate': [0.01, 0.1, 0.3, 0.5],
              'max_depth': [1, 3, 5, 7]}.
```

When a search returned a model with a hyper parameter value being at the edge of our defined grid we expanded the grid, this was repeated until we got hyper parameter values not at the edges. Then we could be sure that we had found a minimum in the hyper parameter space. Finding a minimum (at least locally) in hyper parameter space iteratively in this way may be a time and computationally consuming task, depending on the requirements of the BDT it may be just as well to search for a minimum until one has obtained a model that is precise enough for your purposes. We will see that this is what we have to do with the hyper parameter `n_estimators`.

In the grid above we have $4^3 = 64$ possible combinations of hyper parameter values, which means that `GridSearchCV` will train all 64 possible models and compare them to each other. The search is performed in several stages. First the data sent to `GridSearchCV` will be divided in a number of subsets called *folds*, the default is a 3-fold *cross validation* (CV). For each point in hyper parameter space the method will fit each of the three folds with data, this method ensures that we do not get a result which is overfitted to the data. This means that for the hyper parameter space above, consisting of 64 points, the method has to do 192 fits. Thus, when we have a lot of data and set the `max_depth` to a large number, this search procedure take a lot of time.

The `GridSearchCV` procedure was not a desktop computer task. As it could take several days to do one search we made use of the parallell feature and ran the search on the Abel computing cluster. However, even though we had large computational resources, an exhaustive search of `n_estimators` turned out to consume very much time. Thus we decided to set `n_estimators=500` and tune it manually later. The BDT algorithm is quite robust of overfitting and will therefore always improve for each tree added, thus an exhaustive grid search may give an "unnessecary" high `n_estimators`, *i.e.* we arrive at a model which is indeed good, but it may take up too much disk space. We will discuss how to reduce the file size of a model in Section 4.6.[9]

4.4 Choosing Input Variables for BDT Training

When training a BDT we have to choose which variables to train with. In the case of supersymmetry (or any quantum field theory) we can for example use either the Lagrangian masses or the physical masses. In this project we have tried both and in addition tried to use expressions from the analytical leading order cross sections in Eqs. (2.1-2.2) to simplify the task of the BDT. In the following subsections we will discuss our choices of input variables, the results from the grid searches, and show some of the impact of our choices. The main results are presented and discussed in Section 5.2.

4.4.1 Lagrangian Parameters

First of all, we have to decide which of the Lagrangian parameters to use in the training. As indicated in the paragraph above, we will use the Lagrangian soft masses for the gluino and the first two generations of squarks. The reason for

[9] It is in some `Python` machine learning packages possible to use *early stopping*, which is a function that halts the training procedure when the model improves less than a given limit for a given number of boosting iterations.

Model	Variables used
Lagrangian parameters	M_3, m_{Q_1}, m_{Q_2}, m_{u_R}, m_{d_R}, m_{c_R}, m_{s_R}
Physical masses	$m_{\tilde{g}}$, $m_{\tilde{d}_L}$, $m_{\tilde{d}_R}$, $m_{\tilde{u}_L}$, $m_{\tilde{u}_R}$, $m_{\tilde{s}_L}$, $m_{\tilde{s}_R}$, $m_{\tilde{c}_L}$, $m_{\tilde{c}_R}$
Imitating analytic cross section expression	$m_{\tilde{g}}$, $m_{\tilde{d}_L}$, $m_{\tilde{d}_R}$, $m_{\tilde{u}_L}$, $m_{\tilde{u}_R}$, $m_{\tilde{s}_L}$, $m_{\tilde{s}_R}$, $m_{\tilde{c}_L}$, $m_{\tilde{c}_R}$, $\beta_{\tilde{g}}$, L_2, m_-^2

Table 4.5: Variables used in the training of the three BDTs. $\beta_{\tilde{g}}$, L_2, and m_-^2 are calculated with the mean of the squark masses.

this is that in the analytical leading order cross section expression we only find squark and gluino masses, thus it is reasonable to use the Lagrangian soft masses necessary to construct these physical masses. In Table 4.5 we see an overview of the Lagrangian parameters used. We will now perform a grid search, as discussed in the previous section, with these parameters as input variables and discuss the resulting model.

From a physics point of view we expected a model with large depth and many number of estimators to be the best performing model. In the analytical leading order cross section expression we have only physical parameters, thus it will most likely be harder for the BDT with Lagrangian soft parameters as input to learn the cross section function. The choice of which hyper parameters to include in the grid search was discussed in the previous section, we did a search over the learning rate, depth and number of estimators. However, when the number of estimators grew large the grid search took a very long time, therefore we fixed it to be 500. The grid search result was then:

```
params = {'learning_rate': 0.01, 'loss': 'ls', 'max_depth': 12,
          'n_estimators': 500,'random_state': 42, 'subsample':
          0.5, 'verbose': 1}
```

This result was obtained with a data set with the K-factor problem[10] we discussed in Section 4.1.

We investigate this model by plotting the deviance of the model with the hyper parameters above, see Fig. 4.3. We see that the test set deviance is still decreasing at `n_estimators=500`, thus a longer training will extract more information from the data set. A more thorough discussion of the deviance plots is found in Section 4.5 when we consider the loss functions. The R^2-score for this model is $R^2 = 0.99974$, which is indeed a good score indicating a precise model. However, we must remember that this score is calculated with the logarithm of the cross sections which make it seems better than it actually is.

In Fig. 4.4 we see indeed that the R^2-score is not a good precision measure for us. It is a large skewness in the predictions in our model, small cross sections are underestimated while large cross sections are overestimated. This we could not infer from the R^2-score, and this is why it is important to check the relative deviance of the predictions and the corresponding standard deviation at each decade of the cross section values. To solve this we will train the model more by

[10] We should have repeated the grid search also with the new data. Since the grid searches done in this section were very time consuming we decided against it. The difference between the data sets is only the missing samples with a large K-factor, thus a new search would most likely have yielded a similar result.

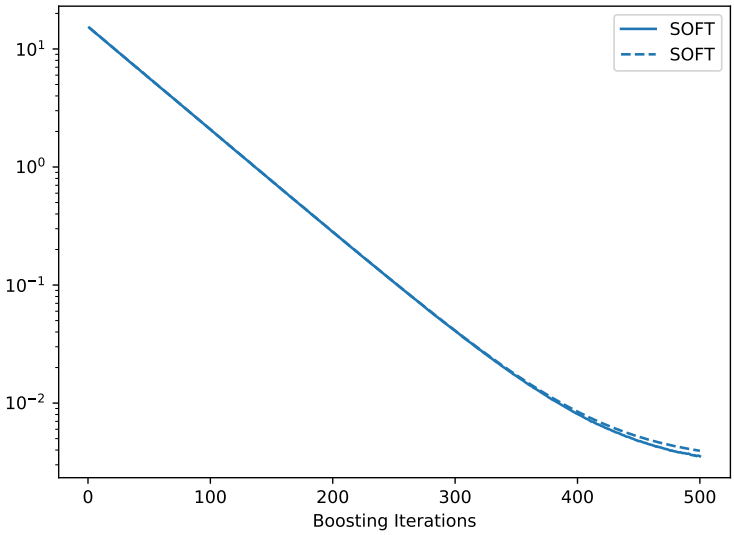


Figure 4.3: Deviance plotted as a function of boosting iterations for BDT trained with Lagrangian soft masses with hyper parameters from grid search.

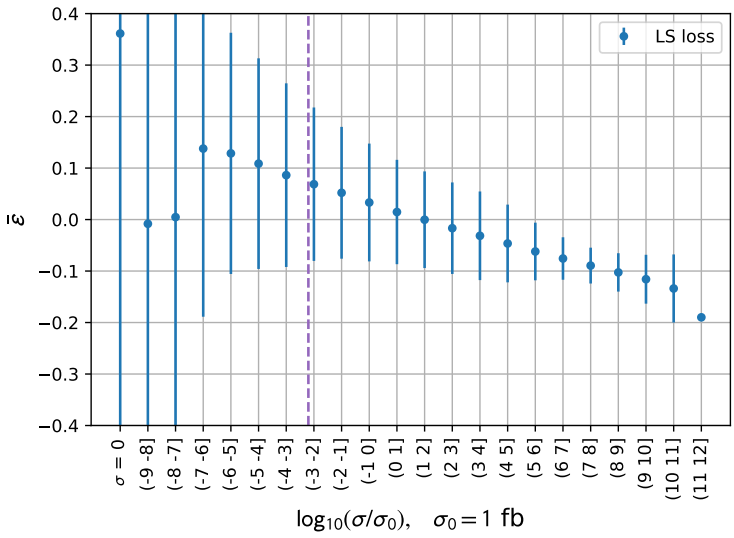


Figure 4.4: Mean of relative deviance for a model trained with Lagrangian soft masses and hyper parameters from the grid search, the error bars show the standard deviation of the relative deviance for each decade. The purple dashed line indicates the cross section for 0.02 events in 20.1 fb^{-1} data at 8 TeV.

increasing `n_estimators` to 1500. We see the result in Fig. 4.5. The skewness is almost not present anymore, which means that the BDT has learned the

cross section function better. However, the standard deviations of the relative deviances are outside our error requirements. For some decades of the cross section values we also have large relative deviances. This may be due to the low number of training samples in these decades of the cross section values.

Considering the hyper parameters used so far and the resulting precision, we may conclude that a model trained with Lagrangian soft parameters is not viable. It also takes $\sim$ 38 hours training the model with `n_estimators=1500` and gives a model using much disk space, a further manual tuning of the hyper parameters will most likely not give a drastic improvement. Thus using an other set of input variables will be the best option to improve the model.

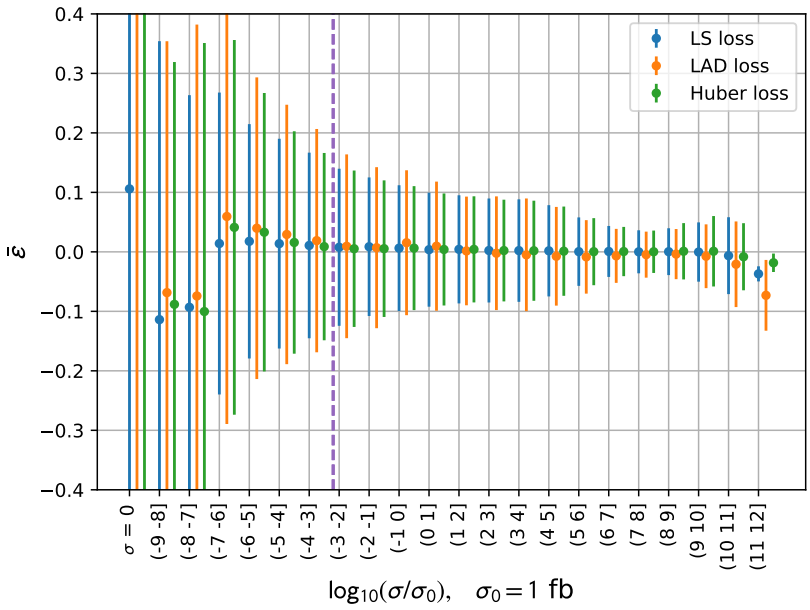


Figure 4.5: Mean of relative deviance per decade for BDTs trained with Lagrangian soft masses and adjusted hyper parameters from the grid search. Only the loss function varies between the models.

4.4.2 Physical Gluino and Squark Masses

We performed similar grid searches over the hyper parameters for a model trained with the physical gluino and first and second generation squark masses. The input variables used are listed in Table 4.5. With these input variables the grid search ended up with the hyper parameter values (also from data with the K-factor problem):

```
params = {'learning_rate': 0.01, 'loss': 'ls', 'max_depth': 13,
          'n_estimators': 500,'random_state': 42, 'subsample':
          0.5, 'verbose': 1}
```

This is almost the same as for the model with Lagrangian soft parameters. The only difference is the depth, which is deeper for this model than for the one

trained with Lagrangian parameters. This is somewhat unexpected, but it may be a sign that this model has learned more of the cross section function than the model trained with Lagrangian soft parameters, *i.e.* the Lagrangian soft model could not figure out the function form even though the depth was increased. Also here we arrived at a model with a large skewness in its predictions, thus we increased the `n_estimators` to 1500 to extract more information from the data set.

As we saw in Fig. 4.5 the model with the Lagrangian parameters did not give us a satisfactory result. Using the physical gluino and squark masses as input variables greatly improved the predictions from the BDT, see Fig. 4.6. Here the mean relative deviance is close to zero for all relevant decades and we also see the standard deviations are below 0.1. Thus we have now a useful predictive model, *i.e.* it is inside our error budget. However, it is possible to push the precision up even more as we will see in the next section. We also note that using the least absolute deviance loss function gives the worst performing model, this will be discussed in more detail in Section 4.5.

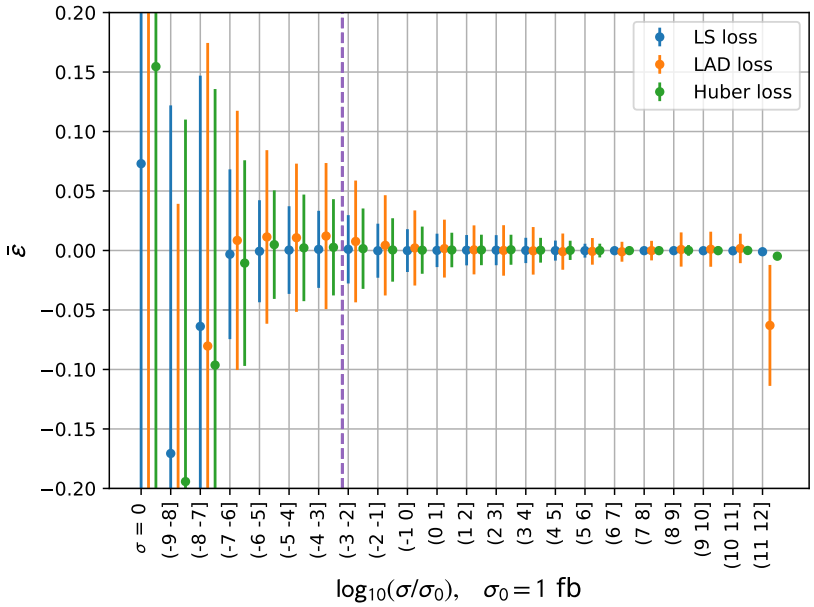


Figure 4.6: Mean of relative deviance plotted as function of decades for a model trained with physical gluino mass, and first and second generation squark masses.

Using all squark masses We expect that the third generation squarks will not affect the cross section for gluino pair production due to the lack of heavy flavours in the proton and flavour-conservation in the scattering process. This can be verified by comparing a model trained with all squark masses and the gluino mass with a model trained with the first and second generation squark masses and the gluino mass. In Fig. 4.7 we see the mean relative deviance for these two models. All hyper parameters are equal,

```
params = {'learning_rate': 0.01, 'loss': 'ls', 'max_depth': 13,
          'n_estimators': 1500,'random_state': 42, 'subsample':
          0.5, 'verbose': 1}
```

As we can see in Fig. 4.7 our model seems to understand the physics. If there had been a change in the precision of the model when adding the third generation squarks we would have a model that have learned something else that is similar to the next-to-leading order cross section. In Section 5.3 we will discuss another test of whether the model has understood the physics.

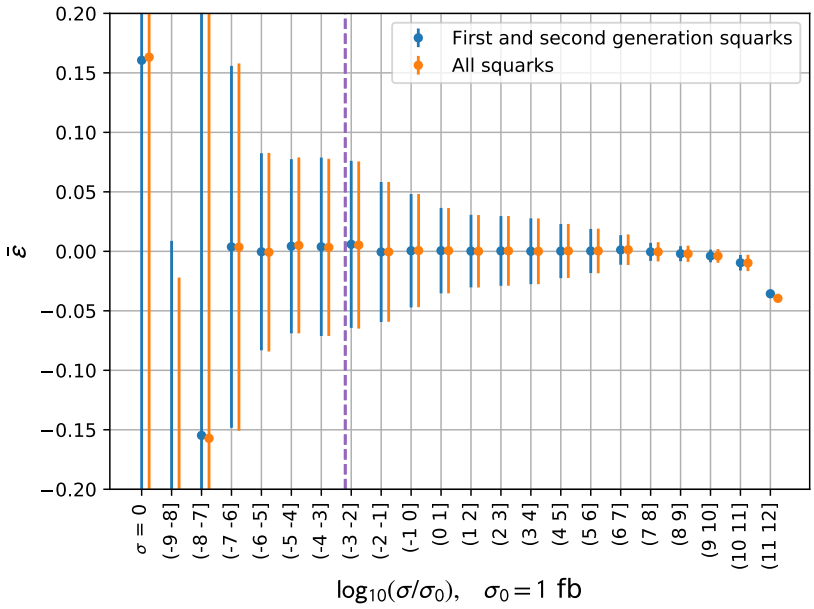


Figure 4.7: Comparison of mean relative deviance for a model trained with the physical gluino mass and first and second generation squark masses, and for a model trained with the gluino mass and all squark masses.

4.4.3 Imitating Analytical Cross Section Expression

We have seen that using the physical masses improves the performance of the BDT compared to BDTs trained with Lagrangian soft parameters. However, we want to investigate further improvements. We will now look at models trained with three additional variables that are used in the analytical leading order partonic cross section expressions for gluino production in Eqs. (2.1-2.2),

$$\beta_{\tilde{g}} = \sqrt{1 - \frac{4m_{\tilde{g}}^2}{s}} \tag{4.3}$$

$$L_2 = \log\left(\frac{s - 2m_-^2 - s\beta_{\tilde{g}}}{s - 2m_-^2 + s\beta_{\tilde{g}}}\right) \tag{4.4}$$

$$m_-^2 = m_{\tilde{g}}^2 - m_{\tilde{q}}^2, \tag{4.5}$$

where $m_{\tilde{g}}$ is the gluino mass, $m_{\tilde{q}}$ the squark masses, and $\sqrt{s}$ the center-of-mass energy. Since the MSSM is a model where the squark masses are in general non-degenerate we can choose between having a m_{-}^2 for each squark or use the mean of the squark masses to find an approximation to m_{-}^2 for each data point. Since each variable added to the model means increased training time, we chose to use the mean of the first and second generation squark masses.

Since we have only added more input variables, we did not do a new grid search. We did however change the hyper parameter `max_leaf_nodes`.[11] Until now we have used the default value which is an unlimited number of leaves. This resulted in models taking much time to train[12] and taking up a lot of disk space, we therefore reduced it to `max_leaf_nodes=100`. We will discuss this more thoroughly in Section 4.6. For now we can state that a model with fewer leaves will in general give poorer predictions than a model with unlimited number of leaves.

Except from the `max_leaf_nodes` hyper parameter, we reused the hyper parameters from the BDT in Section 4.4.2 to check whether it helped to add variables from the leading order partonic cross section expression. The hyper parameters were thus,

```
params = {'learning_rate': 0.01, 'loss': 'ls', 'max_depth': 13,
          'n_estimators': 1500,'random_state': 42, 'subsample':
          0.5, 'verbose': 1, 'max_leaf_nodes': 100}.
```

The results are shown in Fig. 4.8. We see a slight improvement from Fig. 4.6 in mean relative deviance even though we have reduced the number of leaves. However, many decades of the cross section values have larger errors. For the largest cross sections the mean relative deviances are also worse. The increase in standard deviation for the relative deviance is very small, and presumably the information gain from including $\beta_{\tilde{g}}$, m_{-}^2, and L_2 is the reason that the increase was not larger.

It is also important to look at the deviance plot as a check of overtraining and to see whether we would gain more precision by increasing `n_estimators`. In Fig. 4.9 we have plotted the deviance for the three models in Table 4.5 with least squares loss. We do not see signs of overtraining for any of the models. However, the training is not efficient above $\sim$ 500 iterations for any choice of input variables. It seems like all models hits a saturation point here. We also see the benefit of adding more training variables, the test set deviance decrease a factor 6 compared with the BDT model trained with only the physical masses. To deal with the inefficient training we will see that it helps to use other the loss functions.

The slope of the deviance curves tells us how much is learned in each iteration, thus a zero slope on the test set deviance indicates that the model does not learn. The model with Lagrangian soft parameters have the highest deviance scores and is also the only model that seems to not learn much after $\sim$ 500 iterations.[13] The other two have inefficient learning from the same point, however, the slope of the test set deviance is still visibly negative, meaning new features in the data are learned. As long as new features in the data are learned

[11] In Chapter 3 we called it terminal regions.

[12] Of the order $\sim$ 100 hours with only physical masses.

[13] The y-axis is logarithmic, thus the slope may still be slightly negative without being visible at the scale in the plot here.

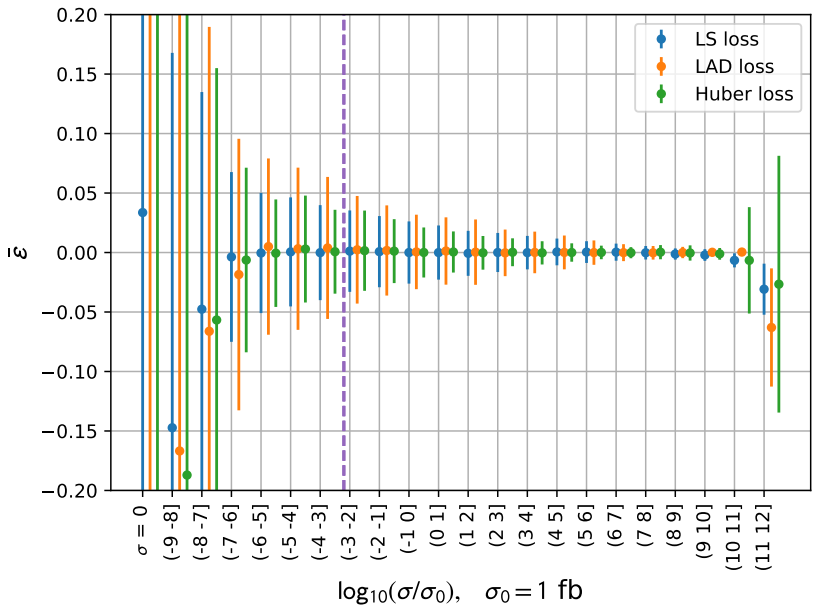


Figure 4.8: Mean relative deviance for models trained with physical the gluino mass, first and second generation squark masses, and $\beta_{\tilde{g}}$, m_{-}^{2}, and L_2 for different choice of loss functions.

we should increase `n_estimators` to extract more information from the training data. We will see in the next chapter that setting `n_estimators=5000` gives a BDT model with very precise predictions.

4.5 Choice of Loss Function

As discussed in Chapter 3 we have to choose a loss function. Based on early tests of each loss function, we found the most promising results for the least squares loss, least absolute deviance loss and Huber loss, the quantile loss was thus discarded in this thesis. In this section we will discuss how each loss function affects training and the resulting BDT model.

When performing the grid searches over the hyper parameters we did not include the loss functions since there are only three viable loss functions available.[14] Thus, tuning this hyper parameter manually is an easy task, *i.e.* comparing three models with all hyper parameters equal except the loss function. We would also check the behaviour of the different loss functions at the different cross section values and how the samples with zero cross sections were handled.

The default loss function for the `GradientBoostingRegressor` is least squares loss. As discussed in Chapter 3, when using a quadratic loss we will be sensitive of outliers in the data set. We have two kinds of outliers in this project, one is the samples with zero cross sections, the other is all other samples that are just

[14] It is, however, possible to implement a user defined loss function in `Scikit-Learn`.

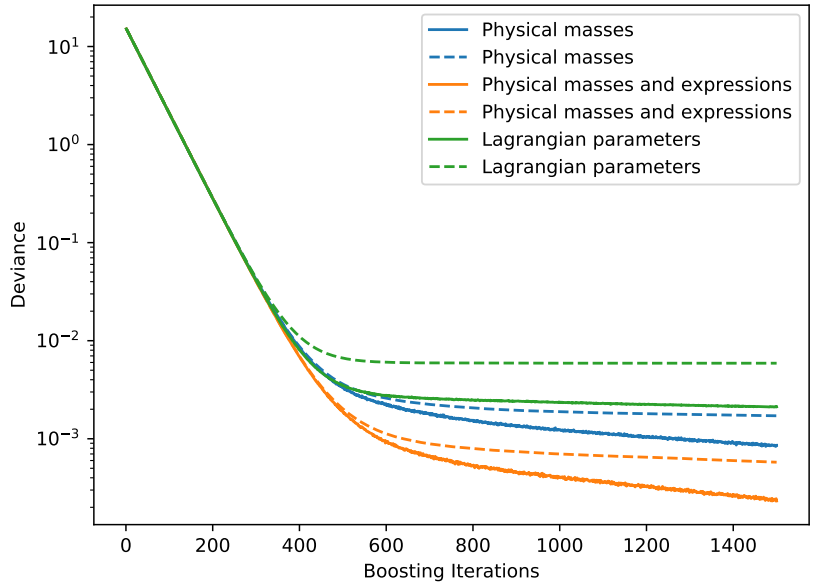


Figure 4.9: Deviance as function of the number of boosting iterations for the models considered in the last three sections with least squares loss. Solid lines are training set deviance and dashed lines are test set deviance.

being predicted poorly.[15] In Section 4.2 we reduced the gap between the zero cross section-outliers and the other samples by setting the cross section values to the power of 10 nearest to and less than the smallest cross section. The reasons for a data point to have zero cross section may be complex for the algorithm to understand, but mostly this is because $m_{\tilde{g}} \geq 4$ TeV (remember that the center-of-mass energy is 8 TeV). Therefore these samples will not behave as the samples with non-zero cross section do and thus they will be harder to learn by the BDT.

The LAD loss is more robust of outliers since the loss is the linear deviance between the target values and predicted values. When calculating the next step in the training the algorithm uses the gradient of the loss function. The gradient of the LAD loss is simply the sign of the residual between the predicted value and target value, thus the algorithm walks in the direction that minimize the loss function with a fixed steplength. (See detailed discussion in section 3.6.) A model using LAD loss may therefore be slower to converge due to a possibly shorter step length than a model with LS loss would have. On the other hand, a BDT using LS loss may take too long steps and can therefore overstep a minimum while the model with LAD loss steps down in to the minimum. If so we will see this in the mean relative deviance plot.

The combination of LS and LAD losses into Huber loss is a compromise between the two loss functions. As we can see from the definition in Eq. (3.16) we have squared loss for samples where $|y_i - \hat{y}_i| \leq \gamma$ (with y_i being the true

[15] Strictly speaking we also have a third category: the samples with sparticle masses outside the chosen intervals, but they were thrown away so we do not have to consider them here.

value and $\hat{y}_i$ the predicted value) and linear loss otherwise.

As an example, in Fig. 4.8 we see (partly) how the samples with zero cross section are treated differently: The BDT using LS loss has $\bar{\epsilon} = 0.032$, while the BDT with LAD loss has $\bar{\epsilon} = 0.56$, and Huber loss has $\bar{\epsilon} = 0.307$. Thus, the model with LS loss works harder to predict the outliers correctly due to its higher sensitivity of outliers, the model with LAD loss gives poorer predictions on all decades of the cross sections, and the model with Huber loss do a better job than the LAD loss model but not as good as the model with LS loss. We can here conclude that the outliers in this data set are easy enough to predict precise without using too much effort on them, if not so the models with LAD and Huber loss would have been more precise than models with LS loss.[16]

To decide on a loss function we use both the mean relative deviance and deviance plots. The model that is not overtrained, has the most efficient training, and has the best precision is the one we want to use. In Fig. 4.9 we saw that the training was inefficient after ~ 500 iterations when using LS loss, independent of the input variables. We will now see that changing the loss function will change this behaviour. We will use the model defined in Section 4.4.3 as an illustration, the behaviour is the same for all sets of input variables used in this thesis.

In Fig. 4.10 we have plotted test and training set deviance for BDTs with the loss functions LS loss, LAD loss, and Huber loss for models trained on physical gluino and squark masses and β_g, L_2, and m^2_- . As noted in Section 3.6 we cannot compare the deviance calulated with different loss functions directly. Therefore we have made an approximation to least squares loss by squaring the deviance calculated with least absolute deviance loss and multiplying the deviance calculated with Huber loss with a factor of two. The deviances of all models start almost at the same place which indicates that we have made a good approximation. However, the test set deviance for the Huber loss model is quite low in the beginning.[17]

For both Huber and LAD loss we have efficient training for all boosting iterations, while for LS loss the training is inefficient after ~ 500 boosting iterations. The test set deviance slope is near zero at 1500 iterations for the LS loss model, while the other two are steeper and may benefit more from longer training. However, not only the training efficiency of the model is affected. The training time also varies with the loss functions. In Table 4.6 we have listed training times and corresponding loss functions. LS loss is the fastest, while LAD and Huber loss uses the same amount of time. Comparing this with the results seen in the mean relative deviance plots so far show that using LS loss may be a good choice even though the training is inefficient. Higher precision on the models with LAD and Huber loss could be obtained with longer training, but it will take longer time than when using LS loss. Thus the LS loss is the best option here.

[16] Outliers like those discussed in Paragraph 4.1 could have disturbed the training of models with LS loss in this way.

[17] We do not know why this happens.

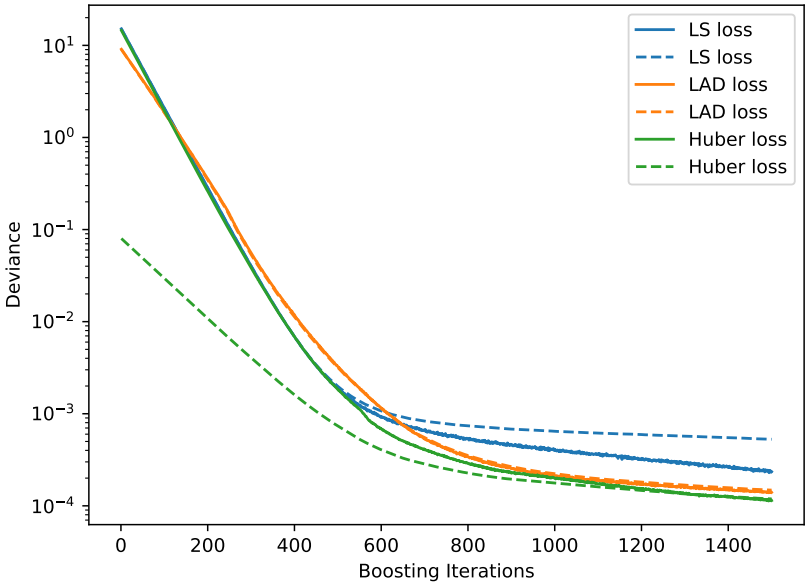


Figure 4.10: Deviance for models trained with physical gluino and squark masses, and β_g, L_2, and m_-^2 where we have used three different loss functions. Solid lines are training set deviances, while dashed lines are test set deviances. The blue lines here are the same as the yellow lines in Fig. 4.9.

Loss function	Time
LS	5.5h
LAD	7h
Huber	7h

Table 4.6: Training times for models trained with physical masses. Hyper parameters equal for all models: `n_estimators=1500`, `learning_rate=0.01`, `subsample=0.5`, `max_leaf_nodes=100`, and `max_depth=13`. The remaining hyper parameters were sat to default values.

4.6 Model pruning

In the previous sections we have discussed our models in the light of which training variables we use and how we adjust hyper parameters such as depth, number of estimators, and learning rate. This gives good results. However, the file that contains the BDT model is very large, up to the order of $\sim$ 800 MB. In this section we will discuss how we can reduce the file size by adjusting the number of leaves and how this impacts the precision of the model.

The file size of a BDT model is important since we must have one BDT for each supersymmetric process. Thus, since there are many possible supersymmetric processes, we have to build many BDTs. Therefore we want each BDT to

be as small as possible in order not to use too much disk space. Another reason a small model is desirable is that the execution time for each cross section value output is proportional to the model size. A smaller model is faster than larger ones.

The size of the model depends obviously on the choice of hyper parameters, however, some hyper parameters are more important than others. The depth and learning rate controls how large the model is, and by adjusting these we affect the resulting prediction heavily. The hyper parameter that control the maximal number of leaves in the tree, `max_leaf_nodes`, may be used to adjust the BDT's tree size and thus file size. It turned out that this parameter can be set to a low value reducing the file size by $\sim 90\%$ and still give good results.

An example of this is shown in Fig. 4.11 where we have plotted three different models. One where we have an unlimited number of leaves, one with 1000 leaves and one with 100 leaves. All models perform well and are well inside our 10% requirement on the standard deviation of the relative deviance. However, the model with 1000 leaves is slightly better at all decades except the two largest cross sections. This is a bit counter intuitive since the model with unlimited number of leaves should have been superior since it can use as many leaves as possible to learn as much as possible. The difference is small, thus this may just be random fluctuations.[18] Using 1000 leaves is not viable any way since the file size of the model will be ~ 200 MB.

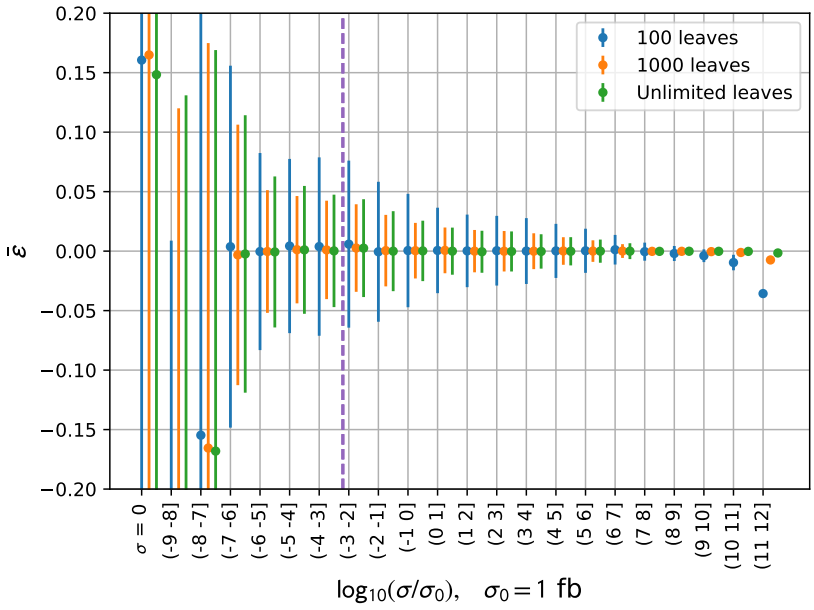


Figure 4.11: Mean relative deviance for models with all equal except the maximal number of leaves.

In Table 4.7 we have listed the time it takes to predict 101085 test samples for different numbers of leaves in a model. It is clear that a small number of

[18] A check for this would be to train models with other random seeds and compare.

leaves indeed improves the speed of the BDT. The difference is small, so it may not be the most crucial parameter to tune.

Number of leaves	Prediction time	File size
100 leaves	5.91s	19 MB
1000 leaves	11.76s	179 MB
Unlimited leaves	16.97s	792 MB

Table 4.7: Prediction times for 101085 test points for different numbers of leaves.

Chapter 5

Results

We will now present our results on building BDT models to evaluate the cross section for gluino pair production at next-to-leading order precision. As discussed in the previous chapter we had two main objectives: to find which input variables and which hyper parameters that gave the best performance. We will first discuss the sufficiency of amount of data used in the training, then look at the performance of the input variables and how they contributed in the learning, then we will also compare the effect of some of the hyper parameters. We finalize with comparing our method to `NLL-fast 2.1` using MSSM-24 data and CMSSM data.

When discussing the different models it may be instructive to see the distribution of data samples versus the decades of the cross section values. We have listed the number of samples in each decade for an example model in Table 5.1. We generated training and test sets by drawing a given fraction of data samples randomly from the total data set, thus the exact number of samples in each decade varies slightly from model to model. See the full discussion of data generation in Section 4.1.

5.1 Learning Curves

Here we will investigate whether the amount of data used to train our models is sufficient. In Fig. 5.1 we have plotted the learning curve for a model trained with physical gluino masses, first and second generation squark masses and $\beta_{\tilde{g}}$, L_2, and m_-^2 with hyper parameters as in Table 5.2. In Section 3.5.3 we discussed the details of how the plot is made.

A first look at Fig. 5.1 suggests that we should have more data, the cross validation score can be improved if so. However, the difference between training and cross validation score is very small, of the order 10^{-4}. Thus we really have enough data. It is important to add that when we train the models considered elsewhere in the thesis we use 90% of the total data set as training data and the remaining 10% as test data, see bottom line of Table 5.1. When plotting the learning curves around 2/3 of the training data is used, thus we will actually have a even slightly better model than in the plot shown here.

Decade	No. of test samples	No. of training samples
$\sigma = 0$	3750	33236
$(-9, -8]$	56	475
$(-8, -7]$	574	5557
$(-7, -6]$	1392	12403
$(-6, -5]$	1903	17165
$(-5, -4]$	2595	22985
$(-4, -3]$	3414	30583
$(-3, -2]$	4356	39453
$(-2, -1]$	5698	52016
$(-1, 0]$	9678	86174
$(0, 1]$	11196	101256
$(1, 2]$	11520	104135
$(2, 3]$	10931	98137
$(3, 4]$	9484	86607
$(4, 5]$	7922	70117
$(5, 6]$	6243	55832
$(6, 7]$	5029	45121
$(7, 8]$	3167	28041
$(8, 9]$	1408	13154
$(9, 10]$	551	5304
$(10, 11]$	216	1930
$(11, 12]$	2	83
Total	101085	909764

Table 5.1: Break down of test and training samples in each decade of cross section values.

5.2 Comparison of Models by Input Variables

The choice of input variable is important, as discussed in Section 4.4. The three choices were Lagrangian soft parameters, physical masses and physical masses with variables from the analytical partonic cross section expression. In this section we will present the three resulting models with their best hyper parameters. It will be clear that choosing physical masses with variables from analytical cross section expression as input variables gives the best performing model.

In Fig. 5.2 we have plotted the mean relative deviance with the best hyper parameter values found for the three sets of input variables used by using `GridSearchCV` and some manual tuning. The hyper parameters are listed in Table 5.2. Note that we did not perform an exhaustive search for the `n_estimators` hyper parameter. When we did the grid search we used `n_estimators=500` which turned out to be too low, in Fig. 4.9 we see that the learning was not finished after 500 boosting iterations, thus we increased it to 1500 as an attempt to exhaust the training data. The hyper parameters will be discussed more thoroughly in the next section.

For the models trained with physical squark masses the standard deviation of ϵ is well inside our requirements down to $\sigma = 10^{-6}$ fb, while the model trained

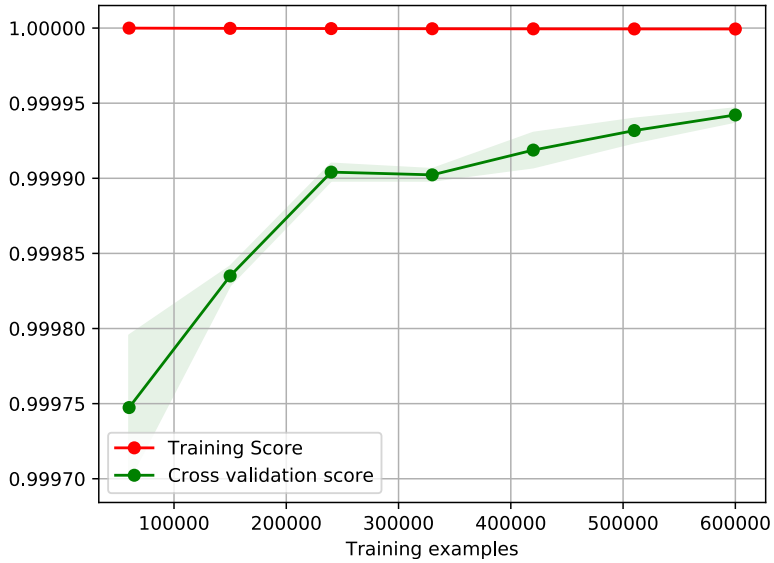


Figure 5.1: Learning curve for a model trained with physical gluino and first and second generation squark masses, and $\beta_{\tilde{g}}$, L_2, and m_-^2. Hyper parameters as in Table 5.2. The y-axis is the R^2-score for cross and training validation.

Input variables → Hyper parameter ↓	Lagrangian	Physical masses	Physical masses and expressions
`'learning_rate'`	0.01	0.01	0.01
`'n_estimators'`	1500	1500	1500
`'random_state'`	42	42	42
`'subsample'`	0.5	0.5	0.5
`'loss'`	`'ls'`	`'ls'`	`'ls'`
`'max_depth'`	12	13	13
`'max_leaf_nodes'`	unlimited	100	100

Table 5.2: Hyper parameters for the three models with different input values. See Section 3.4 for a description of the hyper parameters.

with physical squark masses and expressions from the leading order cross section function has standard deviation of ϵ below 0.1 down to $\sigma = 10^{-7}$ fb. The mean relative deviance is near zero in nearly all decades, except for the two with largest cross sections where we have a slight systematic underestimation of the true values. If we consider Table 5.1 we see that there are few test and training samples in the decades for the largest and smallest cross section values. Thus the BDT has a small amount of data to train with and is expected to perform poorly, in particular when we have limited the maximal number of leaves as discussed in Section 4.6. By comparison, in the decades with many data samples we have good precision.

In Fig. 5.2 we also see the difference when including $\beta_{\tilde{g}}$, m_-^2, and L_2 as training variables. We see a clear improvement for all decades. The drawback by using more input variables is that the training time increases, with the settings

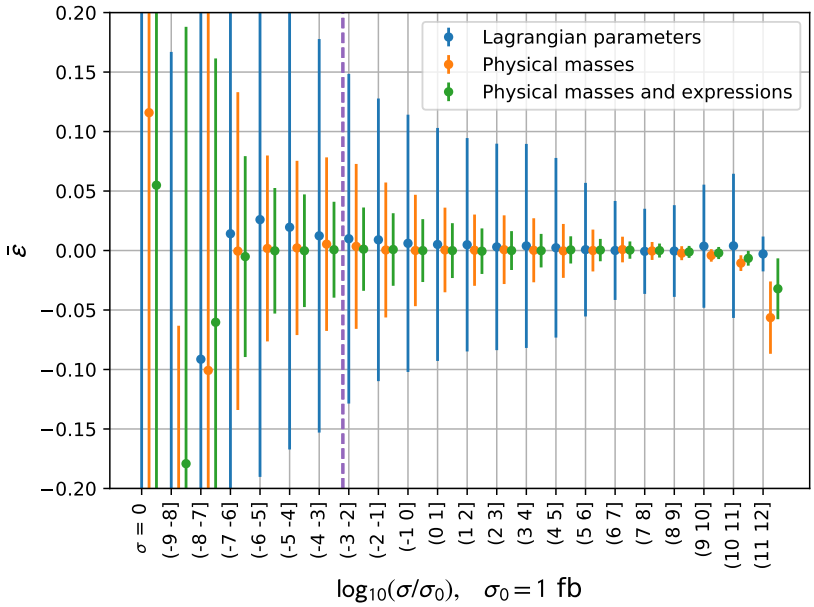


Figure 5.2: Mean relative deviance for each decade of the cross section values for models with Lagrangian parameters, physical masses, and physical masses and variables from the analytical cross section expression. Hyper parameters are listed in Table 5.2. The purple dashed line represents the cross section for 0.02 event in a 20.3 fb^{-1} data set.

in Table 5.2, approximately from 5 hours to 7 hours.

The leftmost points in Fig. 5.2 show the mean relative deviance for the zero-cross section samples. Here the predictions are poor even though we have many data samples. This may come from the fact that these samples are of a complicated nature. There are many ways to get a $\sigma = 0$ data sample, one of the sparticle masses may be too high and there is not enough center-of-mass energy to produce the final state. Thus it may be hard for the BDT to generalize in order to cover all possible cases.

The standard deviation of the relative deviance for the Lagrangian based BDT model is in nearly all decades outside our requirements for how good the precision should be. There may be many reasons for this, the most obvious is that the physical masses are the ones being used in the analytical cross section, thus using Lagrangian parameters will be an approximation. However, for some of the decades for the largest cross section values it is inside our goals of a minimum precision of 10%. This may be due to the low complexity of the cross section function in this area. The gluino mass is low for these cross section values, thus the squark diagrams contributing are suppressed and the cross section function will have a simpler form.

Using the first and second generation physical squark masses together with $\beta_{\tilde{g}}$, m_{-}^2, and L_2 from the analytical partonic cross section expression as input variables has proven to give the most successfull BDT models. In the following we will present results using only use these as input variables.

5.3 Variable Importance

Now that we have found which input variables to use we may do a sanity check of our BDT model, *i.e.* check whether the model has understood the physics. In Section 3.5 we introduced variable importance. For the model trained with physical gluino and squark masses, $\beta_{\tilde{g}}$, m_{-}^2, and L_2 with hyper parameters as in Table 5.3 we expect that the gluino mass will be the most important training variable, while m_{-}^2 and $\beta_{\tilde{g}}$ will contribute significantly since they are quadratically and linearly dependent on the gluino mass.

In Fig. 5.3 we show the variable importance for the BDT. The gluino mass is the most important variable. As expected m_{-}^2 and $\beta_{\tilde{g}}$ are important. And of slightly less importance we have the squark related varibles. They are listed in the order as we expect from the PDF dependence of the cross section, the first generation squarks come first, then the second generation. We also see that $\tilde{u}$ are more important than $\tilde{d}$ since the proton consists of uud. Thus the model seems to have understood the physics involved.

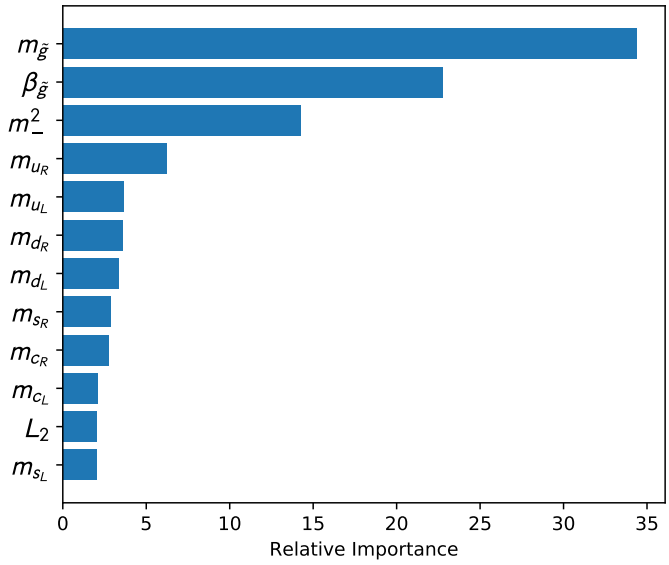


Figure 5.3: Variable importance of BDT with expressions imitating the analytical leading order cross section expression.

5.4 Comparison by Hyper Parameters

In this section we will present our results on the choice of hyper parameters. As discussed in Section 4.3 we used `GridSearchCV` in `Scikit-Learn` to find the best hyper parameters. However, it was necessary to do some additional manual tuning. Due to restrictions on computational power we did not do an exhaustive grid search over the number of boosting iterations. The loss functions were saved

for manual tuning as we mentioned in Section 4.4.1. In Table 5.2 we have the best hyper parameters so far. In the following we will discuss the choice of loss functions and number of boosting iterations, *i.e.* the manual tuning of the hyper parameters.

In Section 4.5 we saw that the models with LS loss had ineffecient learning after $\sim$ 500 iterations. However, inefficient learning is still learning. The deviance plot in Fig. 4.10 shows that the test set deviance is still decreasing at 1500 iterations, therefore a investigation tuning of the `n_estimators` parameter was performed. In Fig. 5.4 we compare $\bar{\epsilon}$ for three models trained with physical squark masses and expressions from the analytical cross section expression. We have set `n_estimators={500,1500,5000}`. Increasing the number of estimators to 5000 does indeed increase the precision of the predictions. Now the systematic underestimation of the large cross sections with `n_estimators=1500` observed in Section 5.2 has also disappeared. Thus a large number of estimators shows further significant improvement. However, the training time has now increased to $\sim$ 24 hours, the model's size increased to 62 MB, compared to around 20 MB when using `n_estimators=1500`. An even larger number of estimators may yield higher precision but at a larger computational cost relative to gained precision.

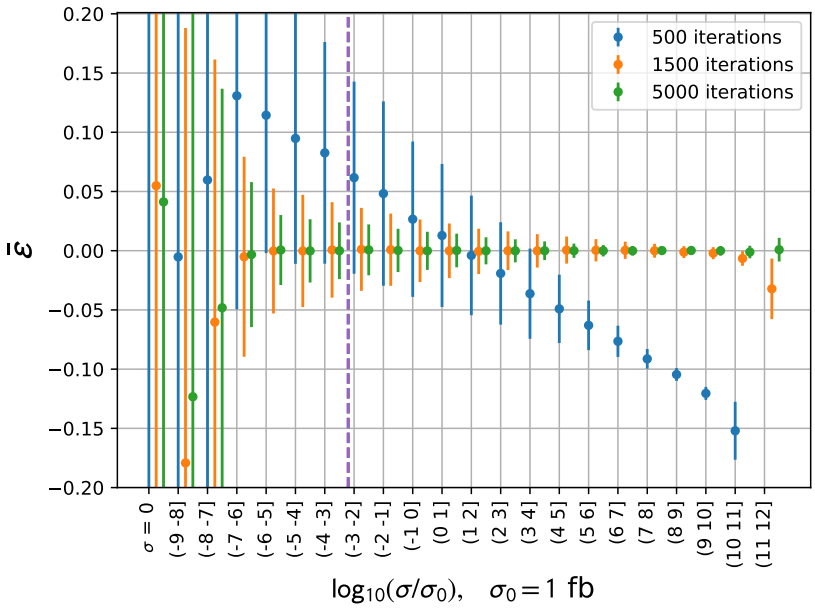


Figure 5.4: Mean relative deviance for three models where only `n_estimators` differs, the other hyper parameters are as in Table 5.2. Input variables used are physical gluino and squark masses together with $\beta_{\tilde{g}}$, m_{-}^2, and L_2.

When we have decided on the number of estimators to be 5000 we may proceed to compare loss functions. In Fig. 5.5 we have plotted the mean relative deviance for three versions of a BDT trained with physical squark masses and $\beta_{\tilde{g}}$, m_{-}^2, and L_2 as input variables. Hyper parameters are listed in Table 5.3, only the loss function differs. We see that the Huber loss function performs slightly

better than LS loss, except in the two decades with the largest cross sections. The LAD loss performs overall poorly compared to the other loss functions. The linear loss do not penalize poor predictions as much as a squared loss does, thus the model will converge slower than a model with squared loss. It is the decades with the smallest and largest cross sections that are most affected by the choice of loss function, this may be due to the low number of data samples for these cross section values. With this in mind we conclude that the best model is the one trained with LS loss.

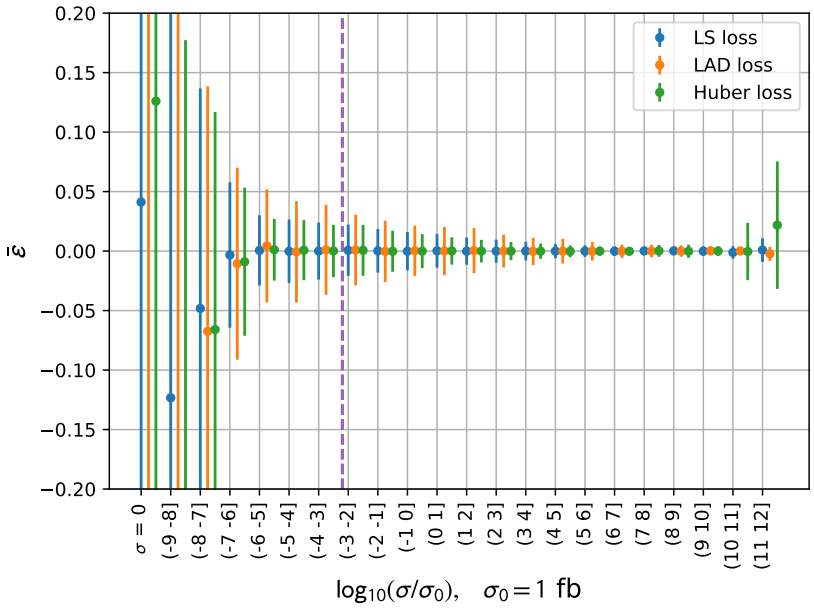


Figure 5.5: Mean relative deviance for BDT models trained with physical squark masses and $\beta_{\tilde{g}}$, m_-^2, and L_2 as input values, with loss function as only hyper parameter differing between the models. The other hyper parameters are listed in Table 5.3.

It is worth noting that higher precision is achievable by increasing the maximal number of leaves. However, the size of the model will be large and will cause practical problems.

5.5 Comparing with NLL-fast 2.1

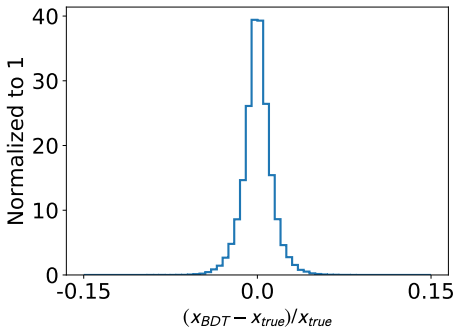
We will now look at the performance of our best BDT model compared to the `NLL-fast 2.1` tool. We have used the BDT trained with physical squark masses and $\beta_{\tilde{g}}$, m_-^2, and L_2, with hyper parameters as specified in Table 5.3. In Fig. 5.6 we have plotted the distribution of the relative deviance for four different cases. The plots to the left show the results from BDT predictions on MSSM-24 and CMSSM data, discussed in Section 4.1, while the results from `NLL-fast 2.1` on the same data sets are shown to the right. Note that the scale of the x-axis varies.

Hyper parameter	Value
'learning_rate'	0.01
'n_estimators'	5000
'random_state'	42
'subsample'	0.5
'loss'	'ls'
'max_depth'	13
'max_leaf_nodes'	100

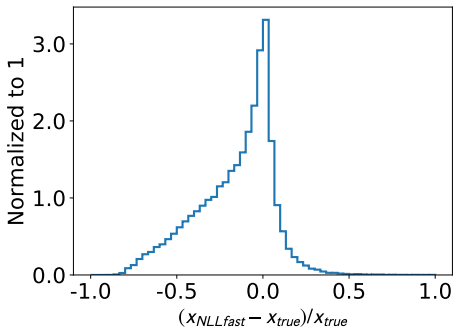
Table 5.3: Final choice of hyper parameters for the model trained with the physical gluino mass, first and second generation squark masses, and $\beta_{\tilde{g}}$, m_{-}^2, and L_2 as input variables.

These distributions show that the BDT does well on both MSSM-24 and CMSSM data. We also see the difference in performance between the BDT and `NLL-fast 2.1` on the MSSM-24 data. The BDT is clearly superior, `NLL-fast 2.1` has a much larger spread and is biased towards low cross sections. The spread in predictions for the BDT is low and the distribution is centered around zero. However, we see a shoulder in the distribution of relative deviance on the CMSSM data. This is due to a bias to larger cross sections for some of the decades of cross sections which is shown in Fig. 5.8. When considering CMSSM data `NLL-fast 2.1` is more precise. That should not come as a surprise since the software is made for the special case of degenerate squark masses. However, the predictions from `NLL-fast 2.1` are slightly biased from the fact that the squark masses in CMSSM are not exactly degenerate due to splitting between $\tilde{q}_L$ and $\tilde{q}_R$ states.

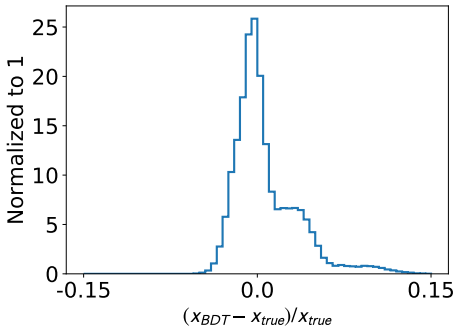
A comparison from a different viewpoint, between our BDT and `NLL-fast 2.1`, can be done by looking at the mean relative deviance for each decade. In Fig. 5.7 we have plotted the results on MSSM-24 data. We see that the BDT clearly outperforms `NLL-fast 2.1` in all decades of cross section values. The reason that the span of cross section values is smaller for these plots are the restrictions on masses given to `NLL-fast 2.1`: the gluino mass must be between 200 GeV and 2500 GeV, and the squark mass between 200 GeV and 4500 GeV. In Fig. 5.8 we see how our BDT performs on CMSSM data compared to `NLL-fast 2.1`. We see that `NLL-fast 2.1` outperforms our BDT method here. However, we again find the slight systematic overestimation, as we saw in Fig. 5.6d discussed above. The BDT here is only trained with MSSM-24 data. If we include CMSSM data in the training, or train a BDT only on CMSSM data, the model will in all likelihood perform better.



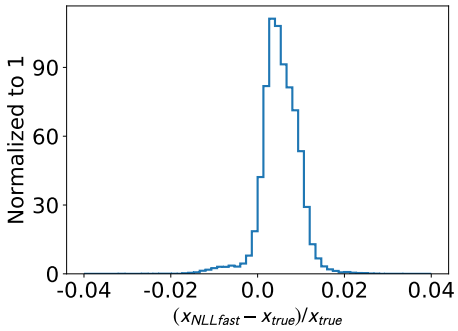
(a) BDT with LS loss function on MSSM-24 data.



(b) `NLL-fast 2.1` on MSSM-24 data.



(c) BDT with LS loss function on CMSSM data.



(d) `NLL-fast 2.1` on CMSSM data.

Figure 5.6: Different predictive models on two datasets.

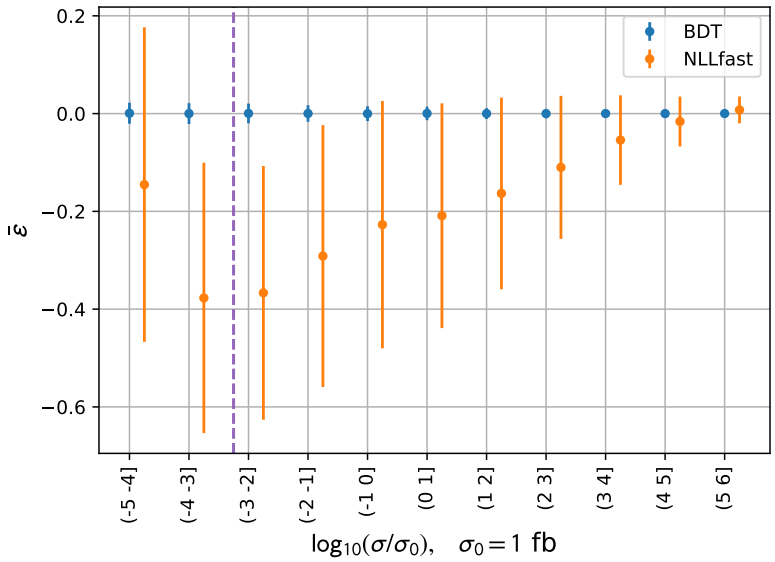


Figure 5.7: BDT model trained with physical masses and expressions from the leading order cross section function compared to the predictions of `NLL-fast 2.1` on MSSM-24 samples.

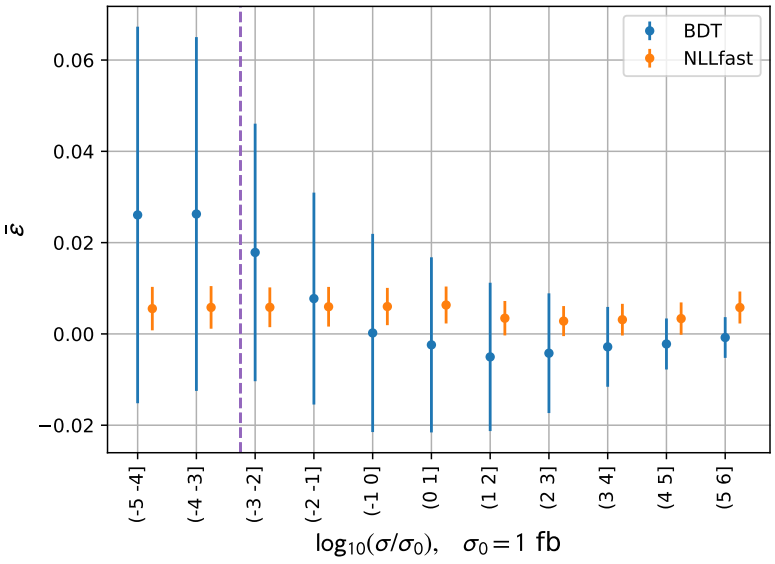


Figure 5.8: BDT model trained with physical masses and expressions from the leading order cross section function compared to the predictions of `NLL-fast 2.1` on CMSSM samples.

Conclusions

The main goal of this thesis was to use machine learning methods to build a fast and precise predictive model for gluino pair production cross sections at next-to-leading-order. When the next-to-leading order cross sections are known and can be evaluated quickly it is possible to do more precise global fits and exclude larger parts of the parameter space of supersymmetric models. In the case of discovery the model parameters can be determined with greater precision. We have indeed successfully built a model with the boosted decision tree algorithm that evaluates the NLO cross section fast for the specified process with a negligible systematic uncertainty and a spread of 10% or smaller, below current contributions from other sources such as higher order terms, the parton density functions and α_s.

The training of the BDT models was done with data generated on the Abel computer cluster. We did quality checks of the data to detect outliers and to see that the parameter space was well covered. Outliers in the data set due to computational problems are important to remove since they will not follow the normal behaviour of the cross section function and thus disturb the machine learning algorithm. We also discovered a bug in `Prospino 2.1` which prohibited K-factors ≥ 10, this lead to many discarded data samples and a gap in our data set. When the bug was corrected we got a data set without gaps in the parameter values.

We used several sets of input variables in order to find the one with best performance. We also performed a hyper parameter scan to find the model with the best precision possible. It turned out that models with a large depth, small learning rate, least squares loss, and many boosting iterations gave the best results. This confirmed our prior views, a complicated function such as the next-to-leading order cross section should lead to a complex predictive model in order to give precise predictions.

To be sure that our predictive model learned the next-to-leading order cross section function and not just remembered the data set we made use of several measures. First we compared the models with the R^2-value defined in Eq. (3.11) when performing the hyper parameter scan. Then we plotted the deviance calculated with the loss function used in the model Eqs. (3.14)-(3.16) to check for overtraining. Finally, the most precise model was found by plotting the mean relative deviance Eq. (3.12) for each decade of the cross section. Our goal was to build a predictive model with standard deviation of the relative deviance smaller than 10%. The results of the best model obtained are shown in Fig. 5.2, where we see that the model with least squares loss the best performance.

Finally, we compared our best BDT model to the state-of-the-arth tool `NLL-fast 2.1`. We found that our method had superior performance on MSSM-

24 parameter points, while `NLLFast 2.1` was most precise on CMSSM parameter points. However, our method is also useful (*i.e.* $\bar{\epsilon}$ is close to, but not precisely, zero and $\sigma_\epsilon < 0.1$) on CMSSM data, but performs less well than `NLL-fast 2.1` for well understood reasons. We also found that our model evaluates NLO cross sections fast. It uses ~ 6 seconds evaluating cross sections from $\sim 10^5$ parameter points.

Further development of this method is to include other supersymmetric processes, for example squark-gluino production. In order to make the resulting predictive model accessible to other physicists a `Python` package (or in a other language, for example `C++`) should be made. We should also implement methods to estimate the error from the PDFs, α_s and re-normalization scale. The result of this thesis may been seen as a "proof of concept" for this.

Bibliography

[1] W. Beenakker, R. Hopker and M. Spira. «PROSPINO: A Program for the production of supersymmetric particles in next-to-leading order QCD». In: (1996). arXiv: hep-ph/9611232 [hep-ph].

[2] W. Beenakker et al. «Squark and gluino production at hadron colliders». In: *Nucl. Phys.* B492 (1997), pp. 51–103. DOI: 10.1016/S0550-3213(97)80027-2.

[3] A. Kulesza and L. Motyka. «Threshold resummation for squark-antisquark and gluino-pair production at the LHC». In: *Phys. Rev. Lett.* 102 (2009), p. 111802. DOI: 10.1103/PhysRevLett.102.111802.

[4] A. Kulesza and L. Motyka. «Soft gluon resummation for the production of gluino-gluino and squark-antisquark pairs at the LHC». In: *Phys. Rev.* D80 (2009), p. 095004. DOI: 10.1103/PhysRevD.80.095004.

[5] Wim Beenakker et al. «Soft-gluon resummation for squark and gluino hadroproduction». In: *JHEP* 12 (2009), p. 041. DOI: 10.1088/1126-6708/2009/12/041.

[6] W. Beenakker et al. «Squark and Gluino Hadroproduction». In: *Int. J. Mod. Phys.* A26 (2011), pp. 2637–2664. DOI: 10.1142/S0217751X11053560.

[7] Emmy Noether. «Invariante Variationsprobleme.» German. In: *Nachr. Ges. Wiss. Göttingen, Math.-Phys. Kl.* (1918), pp. 235–257.

[8] G. S. Guralnik, C. R. Hagen and T. W. B. Kibble. «Global Conservation Laws and Massless Particles». In: *Phys. Rev. Lett.* 13 (1964), pp. 585–587. DOI: 10.1103/PhysRevLett.13.585.

[9] F. Englert and R. Brout. «Broken Symmetry and the Mass of Gauge Vector Mesons». In: *Phys. Rev. Lett.* 13 (1964), pp. 321–323. DOI: 10.1103/PhysRevLett.13.321.

[10] Peter W. Higgs. «Broken Symmetries and the Masses of Gauge Bosons». In: *Phys. Rev. Lett.* 13 (1964), pp. 508–509. DOI: 10.1103/PhysRevLett.13.508.

[11] G. Gabrielse et al. «Erratum: New Determination of the Fine Structure Constant from the Electron g Value and QED [Phys. Rev. Lett. 97, 030802 (2006)]». In: *Phys. Rev. Lett.* 99 (3 July 2007), p. 039902. DOI: 10.1103/PhysRevLett.99.039902.

[12] B. Odom et al. «New Measurement of the Electron Magnetic Moment Using a One-Electron Quantum Cyclotron». In: *Phys. Rev. Lett.* 97 (3 July 2006), p. 030801. DOI: 10.1103/PhysRevLett.97.030801.

[13] Planck Collaboration. «Planck 2015 results - XIII. Cosmological parameters». In: *A&A* 594 (2016), A13. DOI: 10.1051/0004-6361/201525830.

[14] Stephen P. Martin. «A Supersymmetry primer». In: (1997). [Adv. Ser. Direct. High Energy Phys.18,1(1998)]. arXiv: hep-ph/9709356 [hep-ph].

[15] Are Raklev and Paul Batzing. «Supersymmetry Lecture notes for FYS5190». Aug. 2017.

[16] K. Abe et al. «Search for proton decay via $p \to \nu K^+$ using 260 kiloton·year data of Super-Kamiokande». In: *Phys. Rev. D* 90 (7 Oct. 2014), p. 072005. DOI: 10.1103/PhysRevD.90.072005.

[17] Toby Falk, Keith A. Olive and Mark Srednicki. «Heavy sneutrinos as dark matter». In: *Phys. Lett.* B339 (1994), pp. 248–251. DOI: 10.1016/0370-2693(94)90639-4.

[18] Georges Aad et al. «Search for squarks and gluinos with the ATLAS detector in final states with jets and missing transverse momentum using $\sqrt{s} = 8$ TeV proton–proton collision data». In: *JHEP* 09 (2014), p. 176. DOI: 10.1007/JHEP09(2014)176.

[19] Csaba Balázs et al. «ColliderBit: a GAMBIT module for the calculation of high-energy collider observables and likelihoods». In: *Eur. Phys. J.* C77.11 (2017), p. 795. DOI: 10.1140/epjc/s10052-017-5285-8.

[20] Pavel M. Nadolsky et al. «Implications of CTEQ global analysis for collider observables». In: *Phys. Rev.* D78 (2008), p. 013004. DOI: 10.1103/PhysRevD.78.013004.

[21] G. P. Lepage. «A new algorithm for adaptive multidimensional integration». In: *Journal of Computational Physics* 27 (May 1978), pp. 192–203. DOI: 10.1016/0021-9991(78)90004-9.

[22] G. P. Lepage. *VEGAS - an adaptive multi-dimensional integration program.* Tech. rep. CLNS-447. Ithaca, NY: Cornell Univ. Lab. Nucl. Stud., Mar. 1980. URL: http://cds.cern.ch/record/123074.

[23] *Fortran numerical recipes : Vol. 1 : Numerical recipes in Fortran 77 : the art of scientific computing.* eng. Cambridge, 1996.

[24] Warren S. McCulloch and Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259.

[25] Wikipedia contributors. *Artificial neural network — Wikipedia, The Free Encyclopedia.* Online accessed January 25, 2018. 2018. URL: https://en.wikipedia.org/w/index.php?title=Artificial_neural_network&oldid=820907446.

[26] Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The Elements of Statistical Learning.* 2nd ed. Springer Series in Statistics. Springer, 2008. URL: https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12.pdf.

[27] F. Pedregosa et al. «Scikit-learn: Machine Learning in Python». In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[28] Peter J. Huber. «Robust Estimation of a Location Parameter». In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101. ISSN: 00034851. URL: http://www.jstor.org/stable/2238020.

[29] Wes McKinney. «Data Structures for Statistical Computing in Python». In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 51 –56.

[30] Anders Kvellestad. Private communication. 2016.

[31] B. C. Allanach. «SOFTSUSY: a program for calculating supersymmetric spectra». In: *Comput. Phys. Commun.* 143 (2002), pp. 305–331. DOI: 10.1016/S0010-4655(01)00460-X.

[32] Peter Z. Skands et al. «SUSY Les Houches accord: Interfacing SUSY spectrum calculators, decay packages, and event generators». In: *JHEP* 07 (2004), p. 036. DOI: 10.1088/1126-6708/2004/07/036.

[33] Anders Kvellestad. «Chasing SUSY Through Parameter Space». PhD thesis. University of Oslo, 2015.

[34] Wikipedia contributors. *Executable and Linkable Format — Wikipedia, The Free Encyclopedia*. Online accessed January 12, 2018. 2018. URL: https://en.wikipedia.org/w/index.php?title=Executable_and_Linkable_Format&oldid=819857610.