LUND UNIVERSITY
Faculty of Science

Signal and background discrimination for two-electron events in LDMX using a Boosted Decision Tree

Jessamy Mol

Thesis submitted for the degree of Master of Science
Project duration: 10 months

Supervised by Ruth Pöttgen and Torsten Åkesson

Department of Physics
Division of Particle Physics
May, 2019

Signal and background discrimination in two-electron events in LDMX using a Boosted Decision Tree

Jessamy Mol

May 29, 2019

Abstract

The Light Dark Matter Experiment (LDMX) is a fixed target experiment that will search for dark matter, but is still in the development phase. An important aspect for the experiment is the discrimination of signal and background events. Here this signal and background discrimination is inspected using a machine learning technique called a Boosted Decision Tree (BDT). This is done using Monte Carlo simulations of signal and background events, where two electrons hit the target. The signal sample used contains events where one electron undergoes a signal interaction in the target creating a dark matter mediator of a mass of 10 MeV, while the second electron can have any possible interaction (except for a signal interaction). The background sample contains events where one electron loses at least 1.5 GeV of energy by radiating a photon due to a bremsstrahlung interaction. This photon then has a photo-nuclear reaction in the Electromagnetic Calorimeter (ECal), while the second electron can again have any possible interaction. To train the BDT a number of features based on the information from the ECal are used. The distribution of these features is shown for the signal and background samples. The samples are divided into subsets of data, the training data, test data and independent test data. A BDT is trained on the training data and tuned using the performance on the test data. To get a unbiased measure of the performance of the BDT the independent test data is used. The BDT has an Area Under Curve (AUC), for a Receiver Operating Characteristic (ROC) curve, of 0.998981 on the test data and an AUC of 0.998720 for the independent test data. For a background rejection of 99% the signal efficiency is 97% with a BDT threshold of 0.7.

Keywords: Light Dark Matter Experiment, Boosted Decision Tree, machine learning, Dark Matter.

Contents

1 | Introduction

Dark matter has long been a mystery in astronomy as well as in particle physics. When Zwicky [1] was observing galaxy clusters in 1933 he made the discovery that in order to explain the gravitational motion there had to be a large amount of unseen mass. Since this matter does not emit light he referred to it as dark matter. However, the first evidence for dark matter was found as early as 1930 by Lundmark [2].

The building blocks of our universe

The amount of dark matter is actually much larger than the amount of baryonic matter ("normal" matter such as protons and neutrons) which only constitutes 5% of the total mass-energy, while dark matter makes up approximately 27% [3, 4]. The remaining 68% is dark energy, which is thought to accelerate the expansion of the universe. Thus far particle physics can describe baryonic matter with the so-called Standard Model (SM). The particles contained in the Standard Model can be seen in figure 1.1. It is divided into two major groups, the force carriers called bosons, for instance the photon for electromagnetic force, and the fermions. The fermions are again subdivided into quarks, these are the building blocks for protons and neutrons, and leptons, such as the electron.

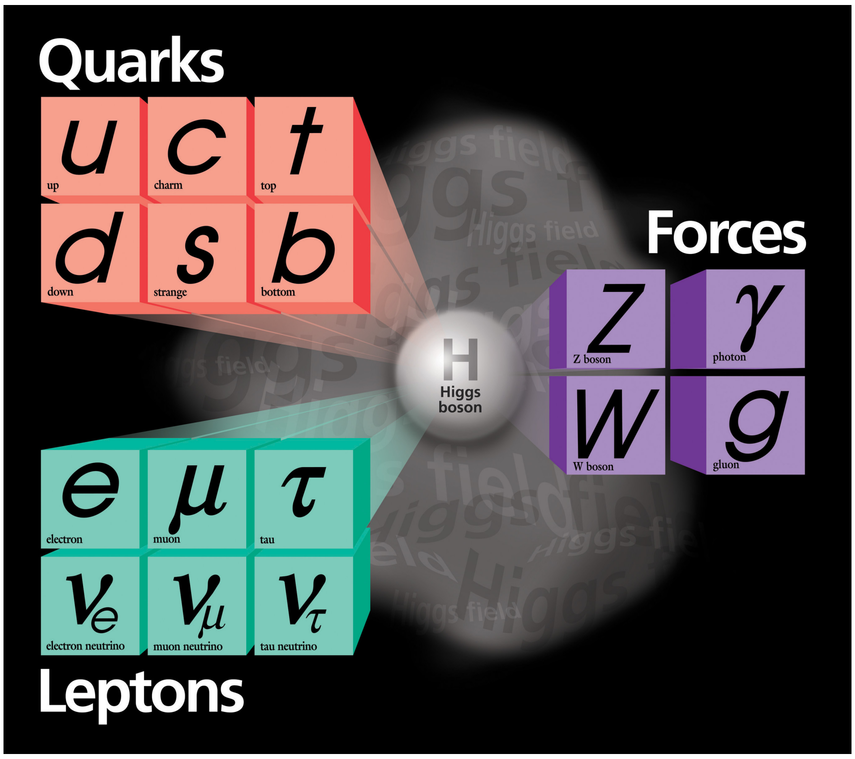


Figure 1.1: The particles in the standard model. Image from [5].

Dark matter does not fit within this model and would be something new. However, dark matter might be a particle or a set of particles which we would be able to discover.

Arguments for the existence of dark matter

Which reasons are there to believe that dark matter actually exists other than the already discussed observations by Zwicky? A number of astrophysical observations have been conducted and have lead to the acceptance of the existence of dark matter. A few of the most important ones will be outlined.

The rotation of stars around the galactic centre has been measured for a number of spiral galaxies. The velocity is expected to decrease as the distance increases. However, the resulting rotational curves look different than expected, they remain fairly flat instead of showing this expected fall. The presence of dark matter would be able to explain the observations [6]. The observed rotational curves as well as an illustration of the observed compared to the expected curve can be seen in figure 1.2.

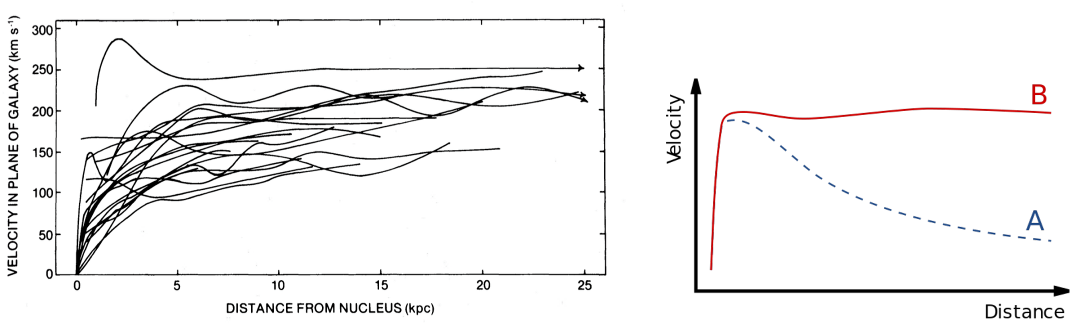


Figure 1.2: On the left the observed rotational curves of 21 spiral galaxies can be seen. On the right an illustration of the expected rotational curve, if there was no dark matter, is shown by the dotted line A, the solid line B shows the observed result. Images from [6] and [7].

Another argument for dark matter is presented by the mass reconstruction of an interacting galaxy cluster, a system with merging subclusters of galaxies, by a technique called gravitational lensing. Very heavy objects can bend the light in their vicinity. When a galaxy is hidden behind such a heavy object, its light will bent around the object and reach the Earth. For us on Earth the hidden galaxy can be seen to appear around the heavy object as, for instance, a ring. How much the light is bent depends on the mass of the massive object. A clear example of gravitational lensing is shown on the left in figure 1.3, the principle of gravitational lensing is shown on the right of the figure.

The mass construction of the interacting cluster 1E 0657-558, containing two merging subclusters, shows that the mass derived from the gravitational lensing effect is much higher than the mass which can be inferred from the emitted light, which indicates the presence of dark matter in the cluster [10].

There are many more observations which prove the presence of dark matter in the universe, such as the comparison of the centre of gravitational mass to the centre of visible mass in the Bullet cluster [11], as well as measurements of the cosmic microwave background radiation by the Planck mission [12].

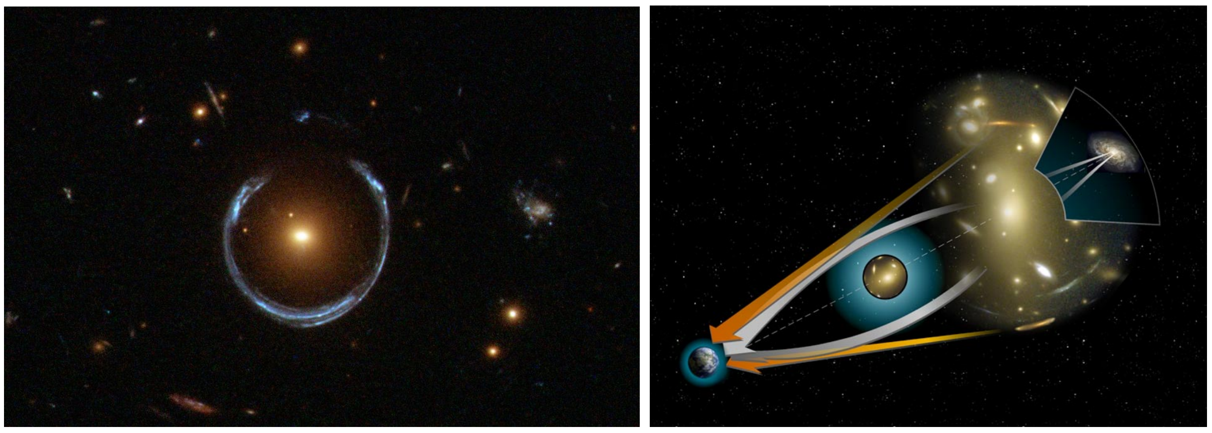


Figure 1.3: On the left an image from the Hubble telescope of a ring due to gravitational lensing is shown. Here the central object causes the galaxy hidden behind it to appear as a ring around it. The image on the right shown an illustration of how gravitational lensing works. Images from [8, 9].

Hypothesis for dark matter particles

There are a number of different candidates for dark matter particles, but all have the general properties expected for dark matter particles [13]:

1. non-baryonic
2. no electric charge
3. no electromagnetic interaction (i.e. no interaction with photons)
4. non-relativistic and therefore not massless
5. stable (or very long lived)

One of the most popular hypothesis is that dark matter consists of Weakly Interacting Massive Particles (WIMPs), which is a general hypothesis that leaves room for different possible particles having other properties [13]. It is a type of thermal relic, where the dark matter is produced in the early universe when the system is in thermal equilibrium. After a "freeze out" the current abundance of dark matter should be left [14]. There has been a focus of research on WIMPs with a mass above $\sim$10 GeV, but nothing has been found so far, causing the exclusion of a large portion of the parameter space [15, 16]. This motivates the search for light dark matter in the sub-GeV mass range, a range in which a lot of the Standard Model particles are present. The specific model which we are interested in, which is a type of thermal relic light dark matter, states that there will be a new mediating particle (like the SM bosons) which would be the connection between the SM particles and the dark matter particles. This mediating particle is called the dark photon and would be able to interact with SM particles, but also decay into dark matter particles [17].

LDMX experiment

The Light Dark Matter Experiment (LDMX) is still in its development phase and will look for dark matter in the MeV-GeV mass range. It will do this using the missing momentum technique, where the momenta of the outgoing particles that have been measured are compared with the momentum of the ingoing particle. A discrepancy between the two indicates an unseen particle. Neutrinos are an example of particles that can be measured using the missing momentum technique. The experiment requires an electron beam which will hit a thin fixed target after which the energy of the (visible) outgoing particles will be measured. The production of dark matter will be similar to bremsstrahlung known from baryonic matter. One type of bremsstrahlung is the production of an energetic photon due to the interaction of a charged particle, in this case an electron, with the electromagnetic field of a nucleus. The dark matter equivalent is called dark bremsstrahlung, here a dark photon (dark matter mediator) is emitted after interaction of the electron with the fixed target [17]. The dark photon is not detected and carries off energy, it can therefore be noticed as missing momentum. See figure 1.4 for the diagram of dark bremsstrahlung.

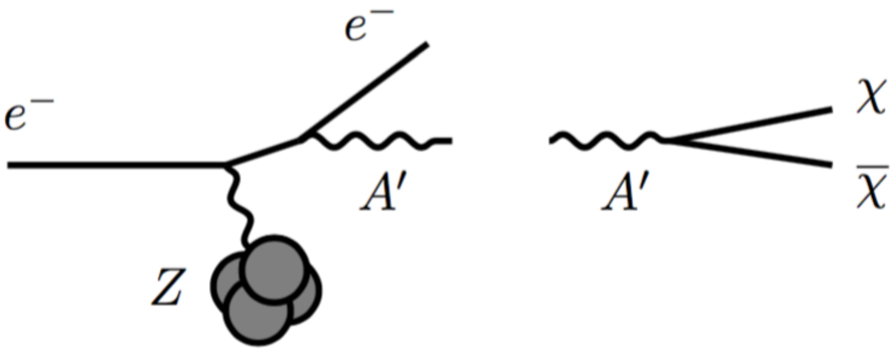


Figure 1.4: On the left the diagram of the radiation of a dark matter mediator (A') from a electron (e^-) due to the interaction with an atom (Z) in the target is shown. The right side shows the successive decay of the dark photon into dark matter particles (χ and $\bar{\chi}$). Image from [14].

A dark matter production signature does not only involve a large energy loss by the 4 GeV beam electron, but also a non-zero transverse momentum of the electron and the lack of other visible final state particles that could explain the energy loss. To be able to have a successful detection of dark matter, all the kinematic components of the electron need to be measured. To do this a recoil tracker and Electromagnetic Calorimeter (ECal) are present, see figure 1.5. The recoil tracker measures the impact position of the electron at the target and the momentum and direction after the target. The ECal, a high granularity Si-W sampling calorimeter consisting of 32 Silicon layers with hexagonal sensors, enables not only the measurement of the total energy deposition, but also the characterisation of showers. The target is made of a thin tungsten foil with a thickness of about 10% of the radiation-length. Before this target a tagging tracker is placed to measure the momentum of the beam electrons to ensure they have the desired 4 GeV and arrive perpendicular to the target. It also measures the position of impact to the target which can be compared to the same measurement by the recoil tracker. It uses a magnetic field of 1.5 T to reject low energy stray particles from the beam halo. Because it is important to measure all visible final state particles that can carry away energy and some of these do not deposit energy in the ECal a second calorimeter is needed. The Hadron Veto Calorimeter (HCal) will identify hadrons not measured in the ECal, such as energetic neutrons from a photo-nuclear reaction in the ECal. The different elements of the LDMX detector are shown in figure 1.5. A more detailed description of the experiment and detector is available in [14] and [17].

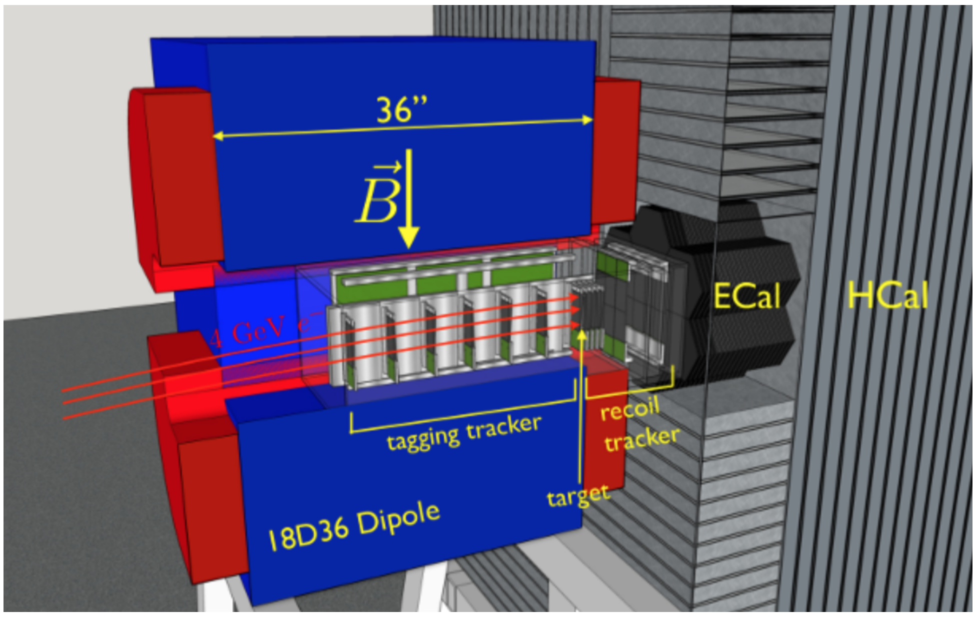


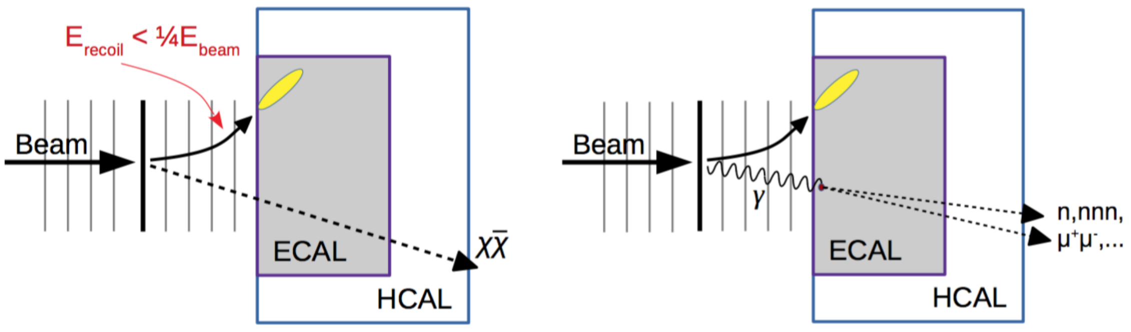
Figure 1.5: Schematic depiction of the LDMX detector design. Image from [14].

In the case of one electron hitting the target and being detected at the time, the selection of interesting events is simpler than when several electrons hit the target at the same time. However, several simulation electrons are needed to be able to get sufficient statistics, even though this will make the selection of a possible dark matter event more complicated. For the optimisation of the LDMX experiment, the study of multi-electron events is therefore an interesting and important topic.

Signal and background discrimination

Since the creation of dark matter would be a very rare event, a lot of events will need to be measured. This creates a lot of data and makes it important to select the interesting events. Most events will be background events, where no dark matter is created, and are therefore not interesting, while a fraction of the events appear to be signal events, where dark matter is created, and merit further investigation. To study the performance and for further development of the LDMX detector, Monte Carlo simulations have previously been performed by the LDMX collaboration of both signal events as well as different types of background events.

Some of the background processes to consider are beam-related background, bremsstrahlung followed by a photo-nuclear interaction or photon conversion to muons and neutrino production in the target. Most of the beam electrons do not interact with the target and just pass through, but this is not a real background since the electron did not lose a lot of energy and therefore does not seem like a signal. However, when a beam electron has a low energy (significantly less than the expected 4 GeV), it will result in a low momentum recoil electron and thus mimic a possible signal. Therefore a tagging tracker is placed before the target, which ensures that all incoming electrons have the desired momentum and any beam-related background events can then easily be removed. The most common interaction of a beam electron with the target is bremsstrahlung. For some of these events the electron will have lost a large amount of energy to the photon. This bremsstrahlung background can be recognised by identifying both the electron and the photon in the ECal. However, if the photon has a photo-nuclear (PN) reaction in the ECal, it will in some cases deposit very little energy and be harder to detect, especially when the reaction products are neutral particles. The HCal is used to identify hadrons produced in the interaction. The photon can also convert to a pair of muons. This process can be recognised by on or two tracks in the ECal. For the scope of LDMX neutrino production processes in the target are so rare that they are negligible. Figure 1.6a shows a schematic depiction of a signal event process, while figure 1.6b shows the depiction of some background processes in the detector.



(a) signal event **(b)** some background events

Figure 1.6: Schematic depiction of a signal event (on the left in 1.6a) and some background processes (on the right in 1.6b) in the detector. Images from [14].

The focus of the research for this project is the use of machine learning to make a differentiation between signal and PN background events. Specifically, a type of supervised machine learning called a Boosted Decision Tree (BDT) will be used. A Boosted Decision Tree has already been studied for events where one electron hits the target [14]. Here the performance of a BDT for signal and background differentiation using Monte Carlo simulations samples where two electrons hit the target will be investigated.

Before using the simulation samples for BDT studies it is important to validate the simulations. It is therefore relevant to plot a number of known distributions, as well as the variables which will be used to train the BDT.

In this thesis we will first look at the Monte Carlo simulations in chapter 2. The application of a BDT for signal and background differentiation will be discussed in chapter 3.

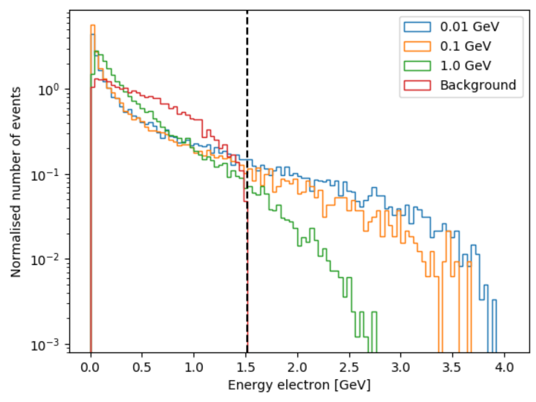
2 | Validation of simulation samples

2.1 Introduction

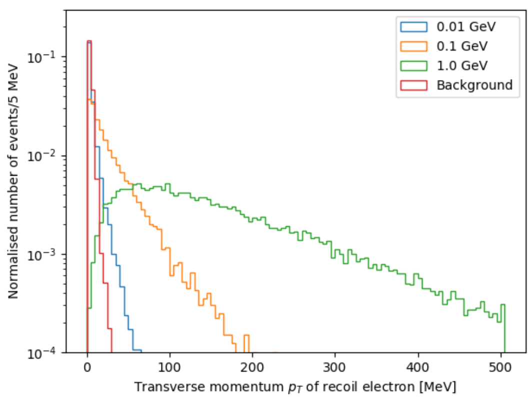
To study the performance of the detector and improve the design, Monte Carlo simulations have previously been performed by the LDMX collaboration. The propagation of particles through the detector and their interaction with the material are simulated with the LIGT framework [14]. This is a custom LDMX interface to the GEANT4 [18] Toolkit. As a closer inspection of the samples resulting from the simulation of signal and background events, some distributions will be plotted. The simulations studied are mostly where two electrons hit the target and some 1-electron samples for reference. In the 2-electron signal samples one electron is required to have a signal interaction, while the other electron is 'inclusive', which means that it can have all possible interactions except for a signal interaction, but will almost always not interact in the target. The 2-electron background samples contain at least one electron which interacts in the target to create bremsstrahlung followed by a photo-nuclear interaction in the ECal, while the second electron is also inclusive. A trigger cut is performed on the 2-electron background simulations samples. It requires that at least one electron has lost a minimum of 2.5 GeV to be able to mimic a signal event, which means that the maximum total energy of the electrons in the simulation events is 5.5 GeV (1.5 + 4.0 GeV). The 2-electron signal samples with a mediator mass of 0.01, 0.1 and 1 GeV contain 72436, 39530 and 77142 events respectively. The 2-electron PN background sample contains 81734 events. For the plots shown in this chapter event weights have been applied for the background samples and all curves are normalised to an area of 1.

2.2 Energy and transverse momentum distribution

We will first look at the energy and transverse momentum distributions of 1-electron and 2-electron samples and compare them, because they are the main discriminators between signal and background events. The energy of the recoil electrons, the electrons after the target, in the 1-electron samples is plotted in figure 2.1a for both signal and PN background samples. The different masses for the different signal samples refer to different hypotheses for the mass of the dark matter mediator A'. The transverse momentum distribution of the recoil electrons is shown in figure 2.1b for 1-electron background and signal samples.



(a) 1e energy



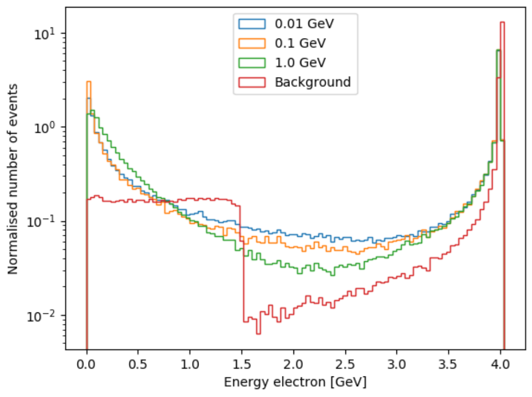
(b) 1e momentum

Figure 2.1: Normalised histogram of the energy (on the left in 2.1a) and the transverse momentum (on the right in 2.1b) of the recoil electron for different dark mediator (A') masses (0.01, 0.1 and 1.0 GeV) and the PN background. These plots are done using 1-electron (1e) samples. The dotted black line in 2.1a is at 1.5 GeV.

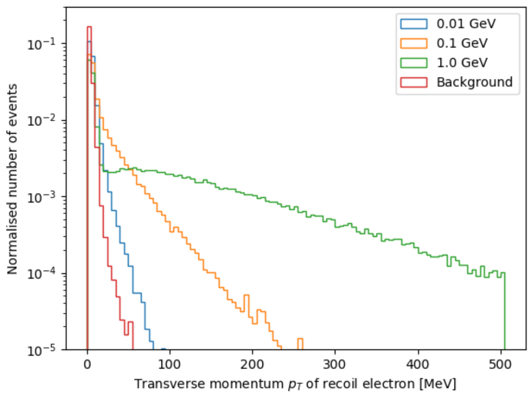
The plots shown in figure 2.1 look similar to those in Fig 10 in [14] and are as expected. When the mediator (A') has a larger mass it will carry away more energy when it is radiated by an electron. The recoil electron will then be left with less energy, this can be seen in figure 2.1a where for the curves with a larger mass the maximum energy is lower. The recoil electron can have a maximum energy of 4 GeV minus the mass of the mediator particle it radiates, for a mediator mass of 1 GeV the maximum energy of the recoil e is 3 GeV. When the mediator mass is higher it can also give a larger "kick" to the electron resulting in a larger maximum transverse momentum, as can be seen in figure 2.1b. The black dotted line in the energy plot (figure 2.1a) is at 1.5 GeV and marks the last bin for the PN background. This is the trigger for the 1-electron background. The recoil electron in the background can have a maximum of 1.5 GeV, anything above is already rejected, because it does not mimic a signal.

The same energy and transverse momentum distribution for the 2-electron signal and PN background samples are shown in figure 2.2, which has an entry for each of the electrons in the event (i.e. two entries per event).

The energy distributions for the 2-electron samples shown in figure 2.2a look different from those for the 1-electron samples (figure 2.1a). The energy distribution curve for the 2-electron signal samples increases for high energies, while for the 1-electron signal samples it decreases with energy. However, for both the 1-electron as the 2-electron samples there is an increase of events for low energies. This part of the 2-electron energy distributions is therefore caused by the electron undergoing a signal interaction. The second electron also has a clear influence on the 2-electron energy distributions. Since this second electron does not undergo a signal interaction, but instead can undergo all other possible interactions, its distribution is different from that of an electron undergoing a signal interaction. The most likely to occur is that the second electron does not interact in the target and maintains its energy of 4 GeV. The most common interaction in the target is that the electron radiates a photon through bremsstrahlung. The probability of radiating a photon with



(a) 2e energy



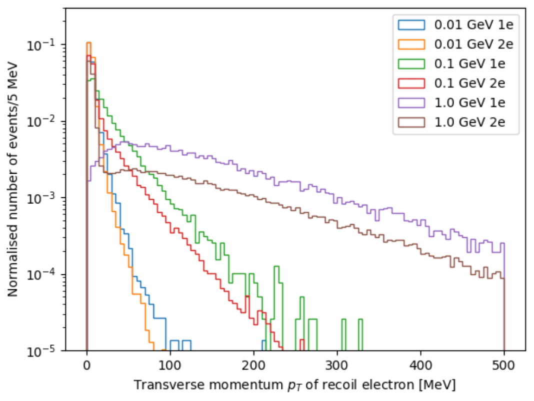
(b) 2e momentum

Figure 2.2: Normalised histogram of the energy (on the left in 2.2a) and the transverse momentum (on the right in 2.2b) of the recoil electron for different dark mediator (A') masses (0.01, 0.1 and 1.0 GeV) and the PN background. These plots are done using 2-electron (2e) samples.

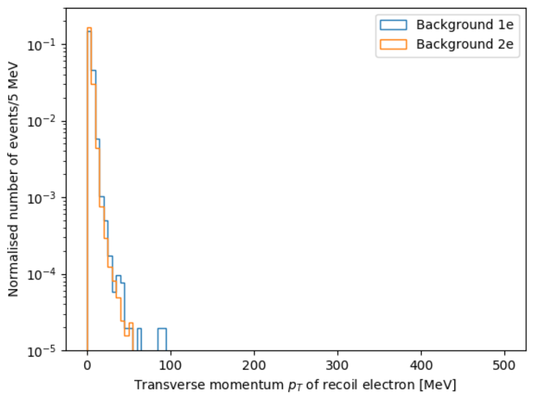
very little energy is much higher that the probability that it radiates a photon with a lot of energy. This means that the energy distribution curve of the second electron increases with energy. The combination of the energy distribution of an electron undergoing a signal interaction and the energy distribution of an electron undergoing background processes therefore creates the distribution curves shown in figure 2.2, where there is an increase of events both for low energies as well as for high energies. The energy distribution for the 2-electron PN background sample seems to show a substructure. This will be looked at later by splitting up the background into different parts.

The distributions for the transverse momentum also differ for the 2-electron and 1-electron signal samples. The 2-electron distributions have more electrons with a low transverse momentum compared to the distributions for the 1-electron samples. To get a more accurate comparison, the transverse momentum distributions for both the 1-electron and 2-electron signal samples are plotted in figure 2.3a.

Based on the most common interactions of the second electron it is most likely to have a small transverse momentum. The addition of the second electron therefore results in a distribution with more electrons with lower values of the transverse momentum. This can most clearly be seen when comparing the distributions for 1-electron and 2-electron signal samples for mediator mass of 1 GeV (see figure 2.3a). The distribution curve for the 1-electron sample decreases at small transverse momenta, while the curve for the 2-electron sample increases for small transverse momenta. The 2-electron distribution curves are all a little below the 1-electron distribution curves for larger transverse momenta. This is because each distribution is individually normalised and the 2-electron curves have additional electrons with a small transverse momentum from the second electron in each event, which means that these distributions will have a lower percentage of electrons with larger transverse momenta. However, for the 2-electron and 1-electron background the distribution of the transverse momentum is very similar, as is shown in figure 2.3b.



(a) 1e and 2e signal momentum



(b) 1e and 2e background momentum

Figure 2.3: Normalised histogram of the transverse momentum of the recoil electron for the 1-electron and 2-electron signal samples (on the left in 2.3a) and PN background samples (on the right in 2.1b).

In the 2-electron PN background sample (shown in figure 2.2) at least one of the electrons produces a direct PN reaction, where the electron radiates a photon (a daughter particle) which has a PN reaction in the ECal. However, it is also possible that both electrons produce a direct PN reaction. The electrons are split up into different classes based on whether they produce a direct PN reaction and on whether the other electron in the event also does this. If both electrons in the event produce a direct PN reaction, then both electrons are classified as `PN both` in figure 2.4 (approximately 9000 events corresponding to $\sim$11%). If only one electron in the event has a daughter which has a PN interaction while the other electron does not, the electrons are divided into different classes (approximately 71000 events corresponding to the other $\sim$89%). The electron that does not produce a direct PN reaction is classified as `non PN`, while the electron which produces the direct PN reaction is classified as `PN once`. The energy distribution for the 2-electron background divided into these three classes is shown in figure 2.4.

The energy distribution for electrons that do not cause a PN through a daughter particle looks as expected (see `non PN` curve in figure 2.4). It decreases as the energy decreases, since it is more likely that the electron loses a little bit of energy rather than a lot of energy. The electron that produces a direct PN reaction while the other electron in the event does not (`PN once` curve) has a maximum energy of 1.5 GeV, like for the 1-electron background, due to the energy requirement in the simulation. This energy requirement also shows in the `PN both` curve where the step at 1.5 indicates that one electron has less than 1.5 GeV while the other electron can have any energy between 0 and 4 GeV.

2.3 Boosted Decision Tree features

Training a Boosted Decision Tree (BDT) requires a number of input variables, so-called features. A BDT has previously been trained on 1-electron data using 10 features. In chapter 3 a BDT will be trained on 2-electron samples using the same features. Here we will first have a look at the distribution of the variables as they are defined in [14], which

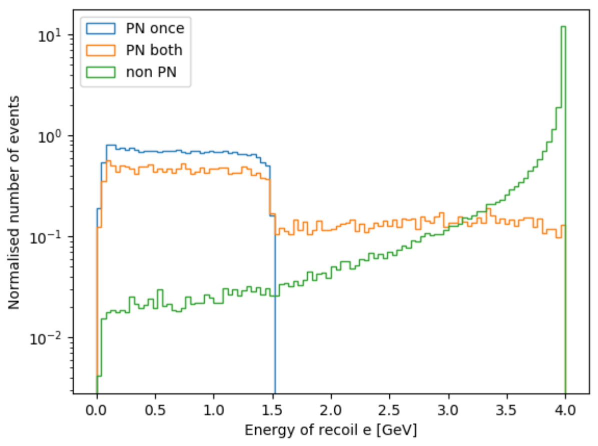


Figure 2.4: Normalised histogram of the energy of the recoil electron for 2-electron background events where the daughter of the recoil e does no PN interaction (non PN) and events where one (PN once) or two (PN both) of the electrons has a daughter which produces a PN reaction.

will later be used as features and the difference between the 1-electron and 2-electron samples.

All the variables used are calculated using the (simulated) data from the ECal. The variables are based on the energy deposition in the ECal as well as the longitudinal and transverse energy distribution in the ECal. Events where the electron misses the ECal are therefore excluded. To determine whether the electron will end up in the ECal, a so-called fiducial region is defined. Any position within 5 mm of an ECal cell centre is defined to be inside the fiducial region, as shown in figure 2.5.

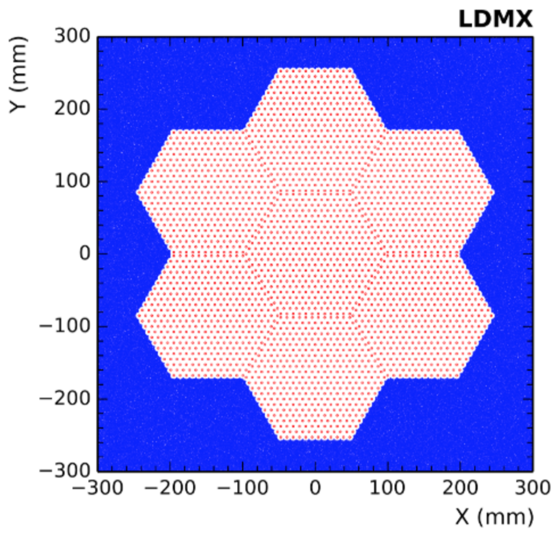
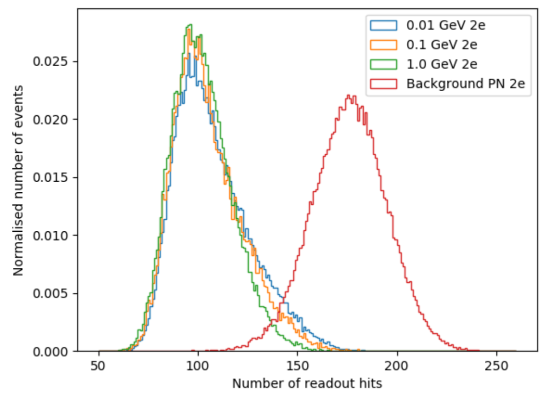


Figure 2.5: The fiducial region covers the area shown in white and red, where the ECal cell centres are indicated by the red dots. The blue area is outside the fiducial region. Image from [14].

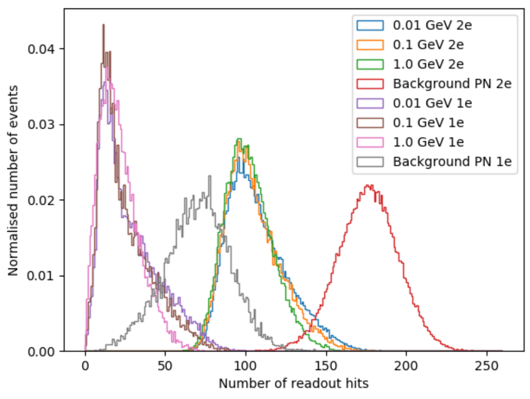
For events where the electron is not within the fiducial region it is not expected to hit the face of the ECal and therefore not used when plotting the distributions. When using the 2-electron samples both electrons are required to be inside the fiducial region.

Number of readout hits

The number of readout hits is defined as the total number of hits in the ECal that are above the readout threshold, which is the energy threshold for reading out an ECal cell. The number of readout hits for 2-electron signal and background samples is shown in 2.6a. For comparison both the 1-electron and 2-electron distributions are plotted in 2.6b.



(a) 2e



(b) 1e and 2e

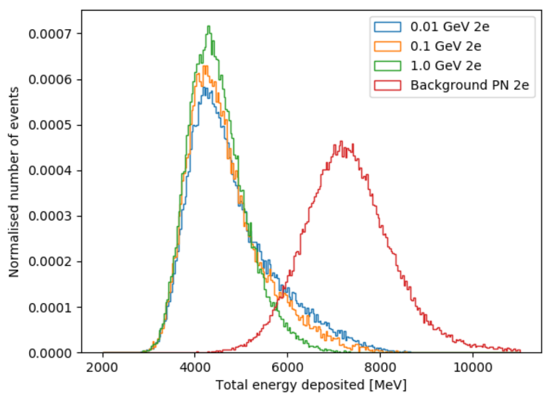
Figure 2.6: Normalised histogram of the total number of readout hits in the ECal for different dark mediator (A') masses and the PN background for events where all of the electrons are within the fiducial region. On the left the distributions are shown for the 2-electron samples, while on the right the distributions for both the 1-electron and 2-electron samples are shown.

There is a good separation between signal and background for the 2-electron distributions of the number of readout hits in the ECal, since only a small fraction of the curves overlap (see figure 2.6a). Compared to the 1-electron distribution, the 2-electron curves are shifted to the right towards a higher number of readout hits, as shown in figure 2.6b. This is because when two electrons enter the ECal instead of one, there are at least two showers which result in more hits. The shapes of the 1-electron and 2-electron curves look fairly similar, but the separation between the signal and background curves appears to be better for the 2-electron samples.

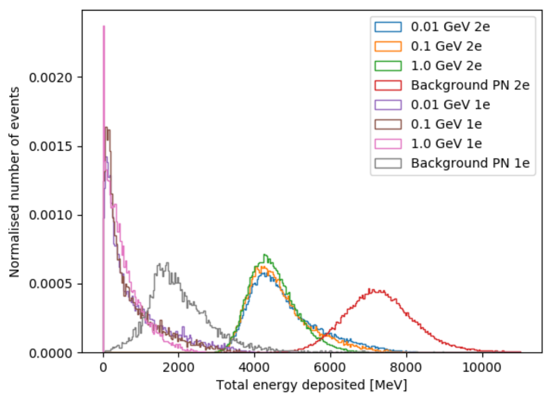
Total energy deposited

The total energy deposited is the sum of the energy of all the hits above the threshold in the ECal. In figure 2.7a the total energy deposited is plotted for 2-electron signal and PN background samples. The 2-electron distributions are plotted together with those for the 1-electron samples in figure 2.7b to be able to compare them.

One striking feature in figure 2.7a is that the energy sum exceeds the expected 8 GeV limit and that the distribution for the background is very different from the distributions for the signal samples (is at much higher energies). Since there are two beam electrons each



(a) 2e



(b) 1e and 2e

Figure 2.7: Normalised histogram of the total energy deposited in the ECal for different dark mediator (A') masses and the PN background for events where all of the electrons are within the fiducial region. On the left the distributions are shown for the 2-electron samples, while on the right the distributions for both the 1-electron and 2-electron samples are shown.

with an energy of 4 GeV, it is not possible to have more than 8 GeV in the ECal. However, the calculation of the total energy deposition in the ECal from the readout energy of the individual cells is based on a calibration. This calibration is not yet fully developed and only done for electrons and not hadrons. It also has a number of assumptions. One of these assumptions is that the particle enters perpendicular to the face of the ECal, which will often not be the case. In the calibration each layer in the ECal is given a weight. These layer weights are given below starting from the weight for the first layer in the front of the ECal and ending with the weight for the last layer at the end of the ECal.

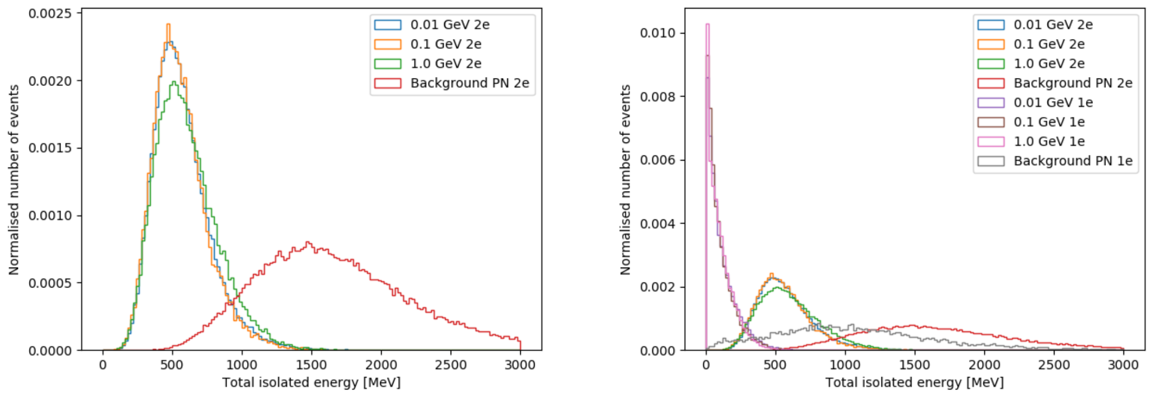
[1.641, 3.526, 5.184, 6.841, 8.222, 8.775, 8.775, 8.775, 8.775, 8.775, 8.775, 8.775, 8.775, 8.775, 8.775, 8.775, 8.775, 8.775, 8.775, 8.775, 8.775, 8.775, 12.642, 16.51, 16.51, 16.51, 16.51, 16.51, 16.51, 16.51, 16.51, 16.51, 16.51, 16.51, 8.45]

The weights towards the back of the ECal are higher than those in the first layers. Since a PN background event will deposit energy deeper in the ECal than signal events (as shown in figures 2.10 and 2.11), these weights could explain the fact that the background curve in 2.7a has higher energies than the signal curves.

Figure 2.7a shows that there is a possibility for discrimination between the signal and background samples, but the signal and background curves still overlap quite a bit. When comparing the 1-electron and 2-electron distributions shown in figure 2.7b, the 2-electron distributions are again shifted to the right towards higher energies. This is because for the 2-electron samples two beam electrons with 4 GeV are used, which means that there is up to 8 GeV of energy rather than just 4 GeV, which is the case for the 1-electron samples. The 1-electron signal distributions are accumulated at zero, but the separation between signal and background is similar for the 2-electron and 1-electron samples.

Total tight isolated energy deposited

The shower centroid is calculated by an energy-weighted average of the (x,y) positions of all hits in the ECal in an event. For 1-electron events this will give the centre of the shower, however, for 2-electron events there are two showers and the shower centroid most likely gives a position between these showers. A ring is then defined to be the neighbouring cells in the same plane as the cell with the shower centroid. The energy of a hit is considered isolated when the neighbouring cells in the same layer are not above the readout threshold. The energy for all isolated hits in the ring is summed up to form the total tight isolated energy deposited in the ECal. Figure 2.8 shows the total tight isolated energy deposited in the ECal for the 2-electron samples as well as both the 1-electron and 2-electrons samples.



(a) 2e **(b)** 1e and 2e

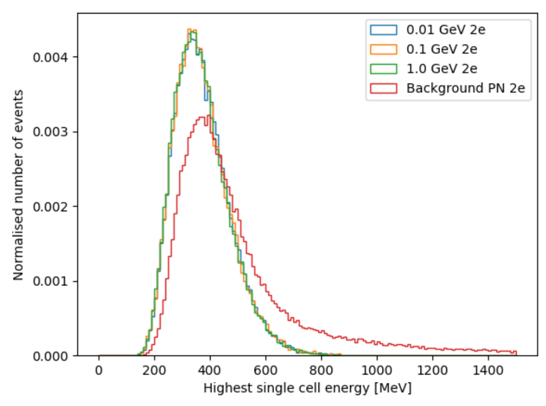
Figure 2.8: Normalised histogram of the total tight isolated energy deposited in the ECal for different dark mediator (A') masses and the PN background for events where all of the electrons are within the fiducial region. On the left the distributions are shown for the 2-electron samples, while on the right the distributions for both the 1-electron and 2-electron samples are shown.

Figure 2.8a shows that there is a decent separation between the signal and background distributions. The 2-electron distributions are again shifted towards higher energies compared with the 1-electron distributions (see figure 2.8b). This is because the shower centroid, around which the total tight isolated energy is calculated, is most likely between the two showers for 2-electron samples in an area at the edge of the showers, where the density of hits is lower, which means that there is more tight isolated energy. There is also more energy coming in to the ECal for the 2-electron samples. The separation between the signal and background distributions seems to be slightly better for the 2-electron samples.

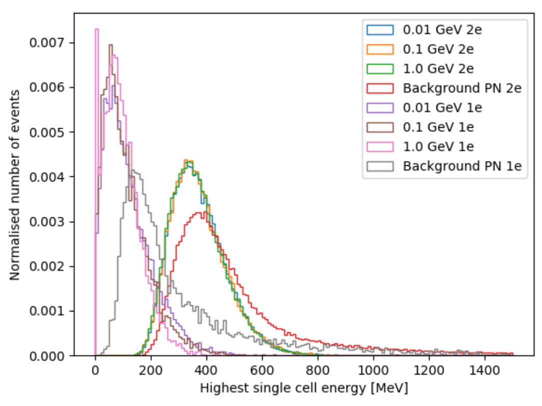
Highest energy in a single cell

The highest energy in a single cell is simply the maximum energy that is deposited in a single ECal cell. The distributions for the 2-electron samples as well as for both the 1-electron and 2-electron samples are shown in figure 2.9.

The separation between the 2-electron signal and background distributions of the highest single cell energy, as shown in figure 2.9a, is very minimal, only for the high energy



(a) 2e



(b) 1e and 2e

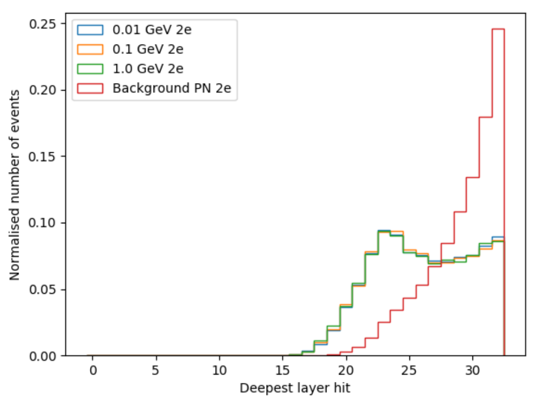
Figure 2.9: Normalised histogram of highest energy in a single cell in the ECal for different dark mediator (A') masses and the PN background for events where all of the electrons are within the fiducial region. On the left the distributions are shown for the 2-electron samples, while on the right the distributions for both the 1-electron and 2-electron samples are shown.

tail a clear distinction can be made (only part of the background distribution). As for the other energy-related variables, the 2-electron distributions are shifted towards higher energies compared to the 1-electron distributions (see figure 2.9b). The separation between the signal and background distributions is somewhat smaller for the 2-electron samples compared to the 1-electron samples.

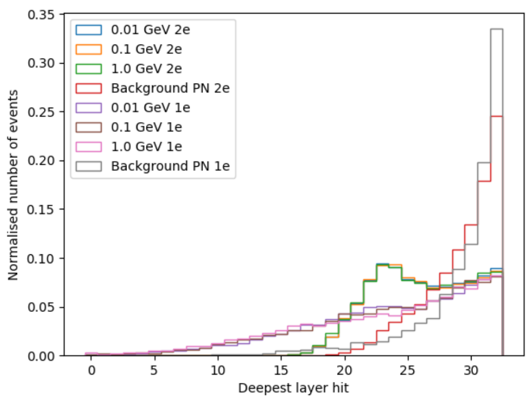
Deepest layer hit

The deepest layer hit is the highest layer number where a hit above the readout threshold has occurred. The variable can have values in the range [0,32] corresponding to the layer number in the simulation, with 0 in front of the ECal and layer number 32 at the end of the ECal. In figure 2.10 the distributions are shown for the 2-electron samples as well as for both the 1-electron and 2-electron samples together.

The PN background has the deepest layer hit mostly at the end of the ECal in the layers 20 to 32 and increases strongly with layer number, while the signal distributions are wider and flatter (see figure 2.10). The distributions for the PN background look similar for 1-electron and 2-electron samples. However, the distributions for the 1-electron signal samples are a lot wider, extending to lower layer number, than the distributions for the 2-electron signal samples. This is because for the 2-electron the distribution is the sum of the distribution of an electron which has a signal interaction, like the 1-electron signal samples, and the second inclusive electron, which most of the time does not interact in the target and will have an energy of 4 GeV or only lose a little energy through bremsstrahlung and will then have an energy close to 4 GeV. This second electron causes the bump around layer 23 in the distributions for the signal samples.



(a) 2e

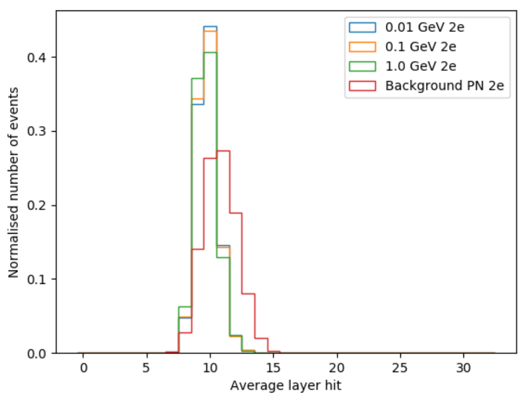


(b) 1e and 2e

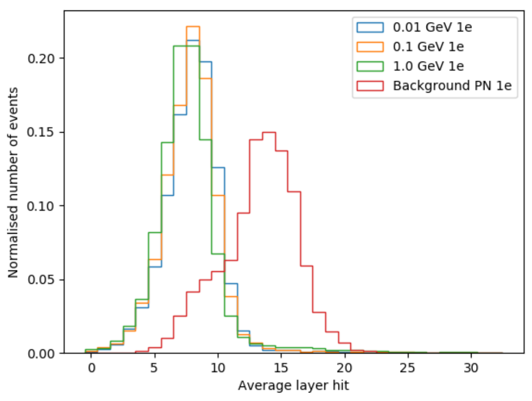
Figure 2.10: Normalised histogram of deepest layer hit in the ECal for different dark mediator (A') masses and the PN background for events where all of the electrons are within the fiducial region. On the left the distributions are shown for the 2-electron samples, while on the right the distributions for both the 1-electron and 2-electron samples are shown.

Average layer hit

The average layer hit is a layer number calculated by taking the energy weighted average of the layer numbers of all the hits above the readout threshold in an event, also in the range [0,32]. Figure 2.11a shows the distributions for the 2-electron samples. The distributions for the 1-electron samples are shown in figure 2.11b.



(a) 2e



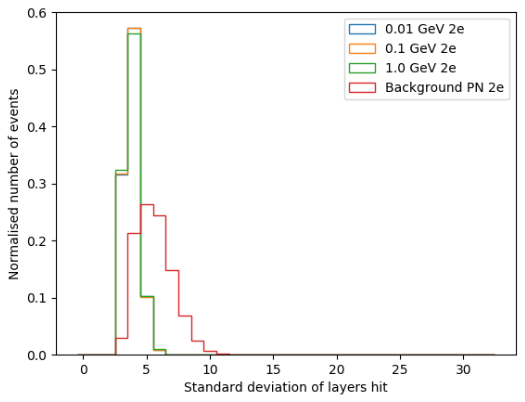
(b) 1e

Figure 2.11: Normalised histogram of the average layer hit in the ECal for different dark mediator (A') masses and the PN background for events where all of the electrons are within the fiducial region. On the left the distributions are shown for the 2-electron samples, while on the right the distributions for the 1-electron samples are shown.

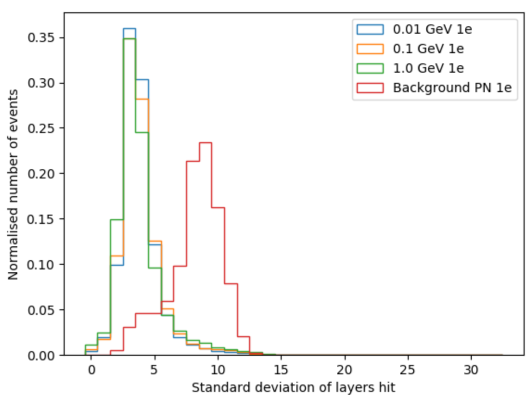
There is almost no separation between the 2-electron signal and background distributions, but the background goes up to a higher average layer hit (see figure 2.11a). The separation between signal and background is much larger for the 1-electron samples, as shown in figure 2.11b. The distributions are also wider for the 1-electron samples. The second inclusive electron is responsible for the fact that the signal and background distributions are much closer together for the 2-electron samples. The electron that has undergone a signal interaction deposits energy mostly in the front of the ECal, having already lost energy. For an event where the electron causes a PN reaction, the energy deposition will be deeper in the ECal. The second electron that has mostly likely had no interaction in the target or has only lost a little energy through bremsstrahlung will deposit energy further in the ECal than the electron which has had a signal interaction, since it most often has an energy of around 4 GeV, which is more than the signal electron. However, it does not have the characteristics of the energy deposit from a PN interaction and will therefore deposit most energy between the single electron signal and background curves, as shown in figure 2.11b, which moves the signal and background curves closer together for the 2-electron samples, as shown in figure 2.11a. Because the average layer hit is energy weighed, this also makes the distributions more narrow.

Standard deviation of layer hits

The standard deviation of layer hits is calculated by taking the standard deviation of the energy weighted layer numbers for all hits in an event. The distributions for the standard deviation of layers hit are shown in figure 2.12a for the 2-electron samples and in figure 2.12b for the 1-electron samples.



(a) 2e



(b) 1e

Figure 2.12: Normalised histogram of standard deviation of layers hit in the ECal for different dark mediator (A') masses and the PN background for events where all of the electrons are within the fiducial region. On the left the distributions are shown for the 2-electron samples, while on the right the distributions for the 1-electron samples are shown.

There is a small separation between the 2-electron signal and background distributions, where the PN background extends to higher layer numbers. Similar to the distributions of the average layer hit, the distributions of the standard deviation of layers hit are more narrow and the distributions of signal and background are closer together for the 2-electron samples (figure 2.12a) compared to the 1-electron samples (figure 2.12b).

Transverse RMS

The transverse RMS is the two-dimensional energy weighted Root Mean Square (RMS) of the shower (or showers, no distinction is made for the case of more than one shower) in the ECal centred on the shower centroid, which is calculated by an energy weighted average of the (x,y) positions of all hits in an event. In figure 2.13 the distributions are shown for the 2-electron samples as well as for the 1-electron and 2-electron samples together.

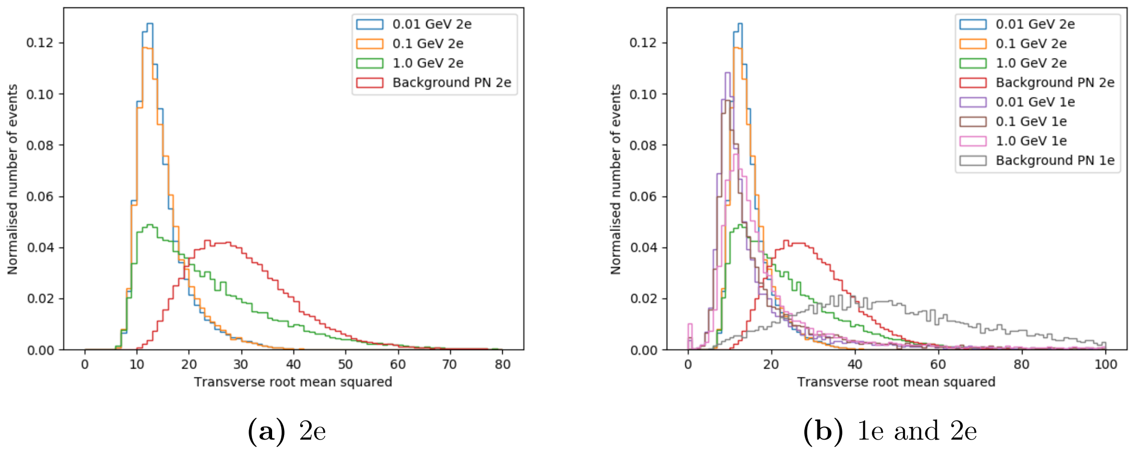


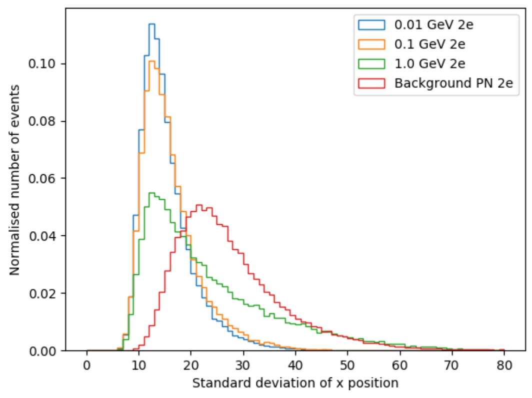
Figure 2.13: Normalised histogram of the transverse RMS for different dark mediator (A') masses and the PN background for events where all of the electrons are within the fiducial region. On the left the distributions are shown for the 2-electron samples, while on the right the distributions for both the 1-electron and 2-electron samples are shown.

The separation between the distributions of the 2-electron signal samples with a dark mediator mass of 0.01 GeV and 0.1 GeV and the PN background is quite decent, as shown in figure 2.13a. However, the distribution for the signal sample with a mediator mass of 1.0 GeV is much wider than for the other signal samples, making it harder to distinguish from the background. The distributions of the 1-electron and 2-electron signal samples look similar. The distribution for the 1-electron background on the other hand is a lot broader than that of the 2-electron background. The distribution for the 2-electron background is also closer to the signal distribution than the 1-electron background distribution.

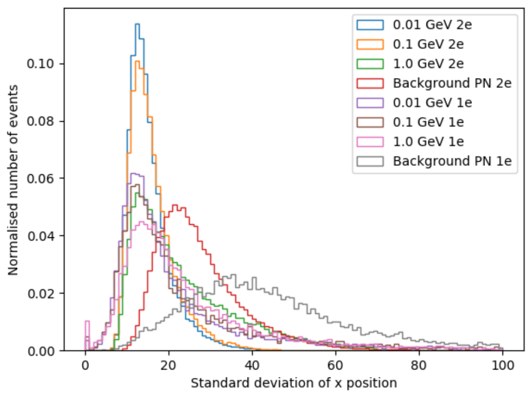
Standard deviation of x and y positions

The standard deviation of the x and y positions is calculated by taking the energy-weighted standard deviations of the x and y positions of all the hits in the event. The distributions of the standard deviation of the x position for 2-electron samples and the distributions for both the 1-electron and 2-electron samples are plotted in figure 2.14. The same plots for the distributions of the standard deviation of the y position are shown in figure 2.15.

The distributions of the standard deviation of the x and y position as well as the transverse RMS all look similar (figures 2.13, 2.14 and 2.15). For all of these distributions the distribution for the 2-electron signal sample with a mediator mass of 1.0 GeV is a lot wider, extending up to higher values, making it harder to separate from the background. For each of the variables the distribution for the 2-electron background is closer to the

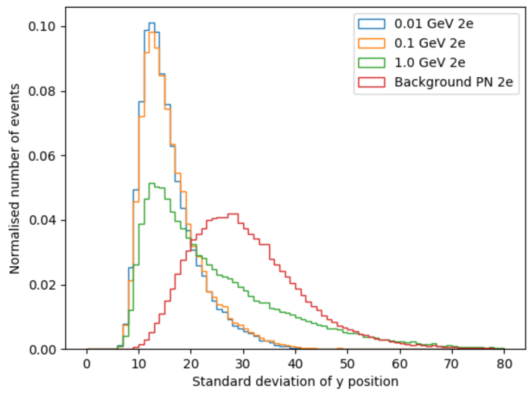


(a) 2e

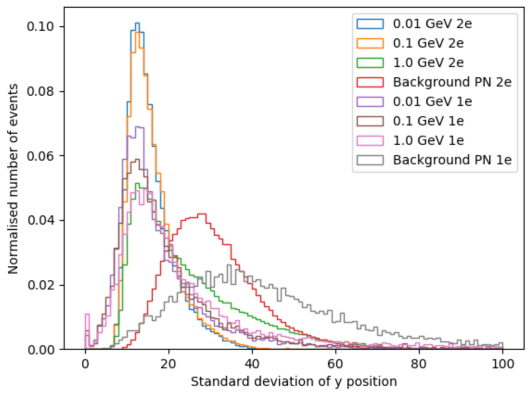


(b) 1e and 2e

Figure 2.14: Normalised histogram of the standard deviation of x position for different dark mediator (A') masses and the PN background for events where all of the electrons are within the fiducial region. On the left the distributions are shown for the 2-electron samples, while on the right the distributions for both the 1-electron and 2-electron samples are shown.



(a) 2e



(b) 1e and 2e

Figure 2.15: Normalised histogram of the standard deviation of y position for different dark mediator (A') masses and the PN background for events where all of the electrons are within the fiducial region. On the left the distributions are shown for the 2-electron samples, while on the right the distributions for both the 1-electron and 2-electron samples are shown.

distributions for the signals and is more narrow than the distribution for the 1-electron background, while the distributions for the signal samples are all similar.

3 | Boosted decision tree

3.1 Introduction

Machine learning plays an important role in the analysis of particle physics data [19]. Here we will look at a type of supervised machine learning called a Boosted Decision Tree (BDT) to distinguish 2-electron signal and background events making use of the Python library XGBoost [20]. For supervised machine learning the computer is provided with both the data as well as labels for this data, these labels are in essence the answer that you want the program to learn. In our case this means that we use a number of variables from the ECal as data (features) and label every event as either a signal or a background event. This data is than split up into two parts, the training sample which will be used for training the BDT and the testing sample which is used to monitor the performance of the BDT for data it has not been trained on.

The classification of events as either signal or background is done based on the different features (variables from the ECal) of the event. The machine learning program first starts with one feature and finds the value of that feature which best separates the signal and background event when the training sample is split into two parts, where the goal is to get as many signal events in one part and as many background events in the other part of the data (somewhat simplified explanation based on [21]). Then it finds an optimal splitting value for every feature and picks the feature with the splitting value that gives the best separation between signal and background events. This is then used to split the events into two so-called branches. This process of improving the separation is repeated to create a tree structure until no further improvement can be made to the purity or a predetermined depth is reached. At this point no further divisions will be made and a leaf is created. In figure 3.1 a schematic representation of an example (from a different experiment) of a decision tree using event hit multiplicity, energy and reconstructed radial position as features for signal and background distinction is shown.

For finding the optimal splitting value and feature, a quantification of how well the separation is for a certain split is needed. In the end we also want to know how well the decision tree classifies signal and background events. To do this, the objective function is defined, which consists of two parts, the training loss L and the regularisation term Ω (see equation 3.1) [22, 23]. This objective function is minimised during training.

$$obj = L + \Omega \tag{3.1}$$

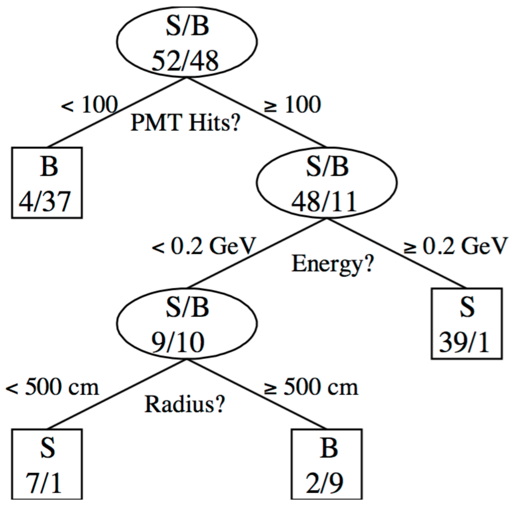


Figure 3.1: Schematic representation of a decision tree showing the tree structure and indicating the signal to background ratios at each point. The leaves are shown by boxes. Image from [21].

The training loss measures how well the predictions from the decision tree (also called the model) fit the desired outcome (labels) on the training data. This means that for each event the model predicts the probability that it is a signal event, this prediction is then compared to the label for that event which is either signal or background. The regularisation term controls the complexity of the model and is used to prevent overtraining, which will be discussed later. An example of a training loss function is the cross entropy error function, also called the logistic loss function,

$$L = \sum_i [y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})] \tag{3.2}$$

where y_i is the label for event i (either $y_i = 1$ for signal or $y_i = 0$ for background) and $\hat{y}_i$ is the prediction for event i [23, 24].

To improve the resulting model, multiple trees can be used in training in a process called boosting. Here the second tree tries to correct the misclassifications by the first tree. This can be done for a predetermined n iterations resulting in a Boosted Decision Tree. Each event is than run through all the trees and assigned a value depending on which leave of the tree it lands in. Taking into account the values it got from the trees, the event is given a final score. For the classification of signal and background, there are only two classes (called binary classification), this score is a probability, i.e. a value between 0 and 1. This can then be transformed into class labels by introducing a threshold value, for instance 0.5, where all events with a score lower than the threshold value get class label 0 (background) and all events with a score above the cutoff value get class label 1 (signal).

The BDT is trained on the training set, but will also need to be able to correctly classify new data, that it has not been trained on, such as the test data. When the model is too simple, both the training and test error will be high and undertraining (or underfitting) occurs. However, when the model is too complex, the test error will be much larger than the training error and overtraining (or overfitting) occurs [25]. An overtrained model learns specific characteristics of the training sample that are not necessarily present in the test data, such as for example statistical fluctuations. This is illustrated for the fitting of a parabolic curve to data points in figure 3.2.

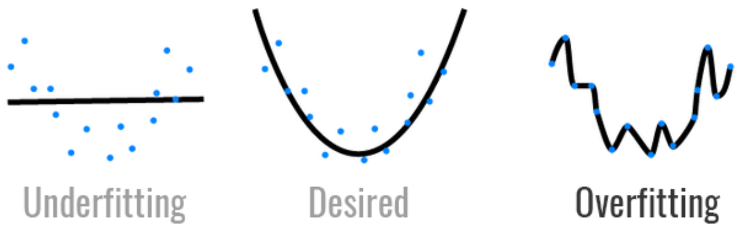


Figure 3.2: A schematic representation of underfitting (left) and overfitting (right) compared to the desired curve (middle) when fitting a parabolic curve to data points. Image from [26].

Whether overfitting occurs depends on the complexity of the model, where for example a BDT containing 100 trees with 20 branches each is more complex than a BDT containing only 5 trees with 3 branches each. The complexity can be controlled by the values of certain variables and regularisation, but also by the number of iterations (number of trees). Figure 3.3 shows the training and test error as a function of the model complexity.

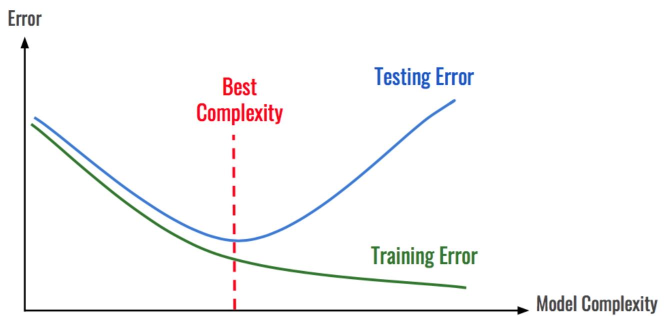


Figure 3.3: A schematic plot of the training and testing error of a machine learning model as a function of the model complexity. The ideal complexity is marked with a red dotted line. Left of the line undertraining occurs while right of this line overtraining occurs. Image from [26].

The best complexity is at the minimal of the test error, before it starts to increase, as shown in figure 3.3. When training the BDT this is when we want to stop adding more trees to the model.

Regularisation is also a way to optimise the performance on the test set and prevent overtraining. One type of regularisation is the L2 norm,

$$\frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{3.3}$$

where λ is a parameter that will be tuned later, T is the number of leaves and w_j is the score on the j^{th} leave [23, 24]. The total regularisation term (part of equation 3.1) also takes into account the number of leaves and is defined as,

$$\Omega = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{3.4}$$

where γ is a parameter that will be tuned later [22].

Another way to prevent overfitting is to add randomness to the model, because this can improve the performance on new data. One possible approach to adding randomness is to only use a certain percentage of the events for the training of a tree in the BDT and randomly selecting these events for each tree. Another method is to only use number of the features for the training of a tree and randomly select which features are used for each tree.

A BDT will be trained using the XGBoost library for python. To get the best performance the best BDT model needs to be found. This is done by tuning a number of BDT parameters. The variables that will be used in the tuning of the model are briefly explained below, where the default values in XGBoost and the total range of the variables are also given (see the XGBoost documentation [20, 27, 28] for more details).

`eta`	This is also called the learning rate and controls the amount of correction done in the next boosting round. It can control how conservative and therefore how complex a model will be. A smaller value leads to a more conservative model. Default = 0.3 Range: [0,1]
`min_child_weight`	The minimum sum of weights needed in a child (a possible leaf on a branch after making a split). This means that no further splits will be made if sum of weights is less than `min_child_weight` and a certain purity has been reached. For larger value the model will be more conservative and this parameter can therefore be used to prevent overfitting. Default = 1 Range: [0,∞]
`max_depth`	The maximum depth of a tree. Larger values lead to a more complex model and overfitting is more likely. Tuning this parameter can therefore be used to prevent overfitting. Default = 6 Range: [1,∞]
`gamma`	The minimum loss reduction required to make a split. A larger value will result in a more conservative model. This parameter corresponds to γ in equation 3.4.

	Default = 0 Range: $[0,\infty]$
`subsample`	The fraction for random sampling of the data. Each iteration a tree will be trained using this randomly drawn fraction of the data (events). This adds randomness to the process and can therefore prevent overfitting. Default = 1 Range: (0,1]
`colsample_bytree`	The fraction for random sampling of the features. Each iteration a tree will be trained using this randomly drawn fraction of the features. Similarly to `subsample` will this parameter also add randomness to the process and can therefore prevent overfitting. Default = 1 Range: (0,1]
`lambda`	Controls the amount of L2 regularisation (for `lambda`=0 no L2 regularisation is present). Increasing this value for the regularisation will make the model more conservative. It corresponds to λ in equations 3.3 and 3.4. Default = 1
`num_boost_round`	The number of boosting iterations. This is equivalent to the number of trees used in the model. Default = 10
`early_stopping_rounds`	When this variable is set (to a value for the number of rounds instead of None) early stopping is used. When the test error does not decrease for the specified rounds the training will stop, no more trees will be added, to prevent overtraining. This is an automatic method for finding the right number of iterations during training (the red dotted line of best complexity in figure 3.3). Default = None
`scale_pos_weight`	When there are a lot more events of one class than the other, unbalanced classes, this parameter can be used for balancing the classes by changing the weights. A typical value is *(number of events of class 0)/(number of events of class 1).* Default = 1

Using these parameters a BDT will be trained to distinguish signal and background events, as described in the next section.

3.2 Method

A BDT will be trained using the Python library XGBoost on the 2-electron Monte Carlo samples, a PN background sample and a signal sample for a dark mediator with a mass of 10 MeV (0.01 GeV). Only events where both of the electrons are inside the fiducial region are used. For the background sample there are 78187 of such events and the signal sample has 56355 of such events. Each event has a value for the 10 ECal features listed below (for plots and description of the features see section 2.3).

0. Number of readout hits
1. Total energy deposited
2. Total tight isolated energy deposited
3. Highest energy in a single cell
4. Transverse RMS
5. Standard deviation of x position
6. Standard deviation of y position
7. Average layer hit
8. Deepest layer hit
9. Standard deviation of layers hit

The early stopping built into XGBoost (`early_stopping_rounds`) will be used to prevent overtraining and to find the best number of iteration. However, this procedure uses the test error which causes the test data to be biased, since it is used in the tuning of the model. To make an unbiased estimate of the performance of the best model that has been found after the tuning procedure, a separate data set is used, the independent test data. To get these three different sets of data the samples are first split into two, where 20% of the events will be used for the independent test and the other 80% is used in the tuning. This 80% is then split into 20% test data and 80% training data.

The distinction between signal and background is a classification problem. The BDT will use logistic regression for this binary classification (the objective) with a logistic loss function (see equation 3.2). For this type of classification the output of the BDT for each event will be a score between 0 and 1. This is the probability that the event is a signal event. This can be transformed to two class labels by using a threshold value where events with a score above the threshold are classified as signal and events with scores below the threshold are classified as background. The objective and a number of other parameters are kept constant for the tuning procedure shown in the list below.

– `objective` = 'binary:logistic'

- `seed` = 5
- `silent` = 1
- `verbosity` = 1
- `early_stopping_rounds` = 10
- `num_boost_round` = 1000 (called `num_round` in the script shown in appendix A)
- `eval_metric`: ['rmse', 'error', 'logloss', 'auc'],

The seed is used for random number generation, for instance for the implementation of `subsample` and `colsample_bytree`. Keeping the seed constant means that the same numbers will be generated every time, that way when running the code several times the resulting BDT will remain the same (as long as all parameters values are kept constant) and not have random variations. By putting both `silent` and `verbosity` to 1 the program will print warnings and errors, but no extra running messages. The parameter `early_stopping_rounds` is kept constant at 10, which means that if the test error does not decrease for 10 rounds an early stop will occur and it will return the performance of the model for the best iteration. To allow the early stopping mechanism to determine the best iteration, `num_boost_round` is set to a high value. Only once the tuning is done and the best model is found will the value of `num_boost_round` be changed to the optimal value. The error metric used to determine the test error used for early stopping is the last item in the list `eval_metric`, which is 'auc'. The 'auc' metric is the Area Under Curve (AUC) for a ROC curve, which plots the true positive rate as a function of the false positive rate. The other metrics in the list are used for comparison. The 'rmse' is the Root Mean Square Error (RMSE), 'error' is the classification error for a threshold of 0.5 and 'logloss' is the negative log-likelihood [27].

A BDT is trained using the script shown in appendix A for a set of values for each of the parameters that require tuning. The tuning is done in several rounds, where in each round for one or two parameters the optimal value is determined. This optimal value is then used in the following tuning rounds. Before the parameters are tuned they are set to the default value from XGBoost. For each tuning round a submission script is written which runs the BDT script (appendix A) multiple times. An example of such a submission script is shown in appendix B. The ranges of values for which the parameters are tuned is shown below for each round.

1. (a) `eta`: [0.01, 0.05, 0.1, 0.2, 0.3]
   (b) `eta`: [0.02, 0.03, 0.04, 0.06, 0.07, 0.08, 0.09]
2. (a) `min_child_weight`: [1, 3, 5, 7, 9] and `max_depth`: [3, 5, 7, 9]
   (b) `min_child_weight`: [6, 7, 8] and `max_depth`: [1, 2, 3]
3. `gamma`: [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
4. (a) `subsample`: [0.5, 0.6, 0.7, 0.8, 0.9, 1.0] and `colsample_bytree`: [0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

   (b) `subsample`: [0.85, 0.95] and `colsample_bytree`: [0.60, 0.7, 0.8]

5. (a) `lambda`: [0.0, 0.3, 0.7, 1.0, 1.5, 2, 5]
   (b) `lambda`: [0.8, 0.9, 1.1, 1.2, 1.3, 1.4, 8, 10, 15]
   (c) `lambda`: [13, 14, 16, 17, 18, 19, 20, 25, 30, 40, 50, 100]
   (d) `lambda`: [32, 34, 36, 38, 39, 41, 42, 44, 46, 48, 500, 1000]
   (e) `lambda`: [45, 47]
6. `scale_pos_weight`: [True, False]

Each parameter is tuned using an initial range of values. If, based on the result of that first tuning round, the range of values needs to be expanded to determine the optimal value more accurately, a second tuning round is done for that parameter with additional values around the optimal value found in the initial tuning of the parameter (round b in the list above). To determine the optimal value the AUC (of the ROC curve) for the test set is used, where the value for the parameter with the highest AUC is considered optimal. A number of parameters are tuned together in the same round. This is done to get the best combined values for those parameters. Ideally all parameters would be tuned in a single round, because the order in which the parameters are tuned has an influence on which optimal value is found for the parameters. However, not all parameters are tuned simultaneously, since the number of combinations for the values of the parameters would be so large that it would take too long to get a tuned model. Therefore a maximum of two parameters are tuned in the same round. The parameters `min_child_weight` and `max_depth` are tuned together because both these parameters control the complexity of the model. Since `subsample` and `colsample_bytree` are the parameters to introduce randomness into the model these are also tuned simultaneously.

When the optimal values for the parameters have been found, the best model using these optimal values is trained. The performance of the BDT will than be evaluated on new data, the independent test set, that has not been used in the tuning procedure and is therefore unbiased. The error metric used for the evaluation of the performance of the BDT is AUC (for the ROC curve), which is also the metric used to determine the optimal values in the tuning of the model as well as for the early stopping, where a higher AUC is a better performance.

3.3 Results

The different values for `eta` used in the tuning procedure together with the AUC for the test set for the resulting BDT are shown in table 3.1a, where the results for both round 1a and 1b are shown (see section 3.2). The values of the other parameters are kept constant, while `eta` is tuned and are shown in table 3.1b. Before these values are tuned they are set to the default value (as defined in XGBoost).

The optimal value of `eta` from the tuning results (see table 3.1) is 0.08. For this value of `eta` the AUC for the test set is 0.99892, while for the default value of `eta` it is 0.998856.

Table 3.1: Tuning results for `eta`. In (a) the different values of `eta` are shown with the AUC of the resulting BDT. The values in bold indicate the best values resulting from the tuning. In (b) the values of the other parameters that are kept constant during this tuning round are shown.

(a)

`eta`	AUC
0.01	0.99872
0.02	0.998886
0.03	0.998917
0.04	0.998897
0.05	0.998904
0.06	0.99891
0.07	0.998912
0.08	**0.99892**
0.09	0.998899
0.1	0.998902
0.2	0.998886
0.3	0.998856

(b)

`max_depth`	=	6
`min_child_weight`	=	1
`gamma`	=	0
`subsample`	=	1
`colsample_bytree`	=	1
`lambda`	=	1

The results, of both round 2a and b (see section 3.2), of the simultaneous tuning of `max_depth` and `min_child_weight` are shown in table 3.2a. The values for the other parameters that are kept constant are shown in table 3.2b, now with the tuned value for `eta`.

The optimal values resulting from the tuning (shown in table 3.2) are `max_depth` = 7 and `min_child_weight` = 2. Tuning these parameters increases the AUC slightly to 0.998931.

The results of the tuning of `gamma` are shown in table 3.3a. The values of the other parameters are shown in table 3.3b, with the tuned values for `eta`, `max_depth` and `min_child_weight`.

The optimal value of `gamma` is found to be the default value of 0 as shown in table 3.3. This means that the AUC is not increased in this tuning round.

The results of the simultaneous tuning of `subsample` and `colsample_bytree` are shown in table 3.4. Where the initial tuning round (4a, see section 3.2) is shown in table 3.4a and the second round (4b, see section 3.2) is shown in table 3.4b. The values of the other parameters are given in table 3.4c.

The optimal values resulting from the tuning are `subsample` = 0.9 and `colsample_bytree` = 0.7 (see table 3.4). The tuning of these parameters improves the AUC slightly to a value of 0.998951. Since there are only 10 features `colsample_bytree`, is tuned to an accuracy of one decimal. The second tuning round (shown in table 3.4b) attempts to tune `subsample` more accurately, but no better value than the one resulting from the first tuning round is found.

The tuning results for `lambda` are shown in table 3.5. The first three tuning rounds (a-c in

Table 3.2: Tuning results for `max_depth` and `min_child_weight`. In (a) the different values of `max_depth` and `min_child_weight` are shown with the AUC of the resulting BDT. The values in bold indicate the best values resulting from the tuning. In (b) the values of the other parameters that are kept constant during this tuning round are shown.

(a)

`max_depth`	`min_child_weight`	AUC
3	1	0.99884
3	3	0.998839
3	5	0.998842
3	7	0.998839
3	9	0.998838
5	1	0.998892
5	3	0.998871
5	5	0.998884
5	7	0.998898
5	9	0.998905
6	1	0.99892
6	2	0.998901
6	3	0.998894
7	1	0.998927
7	**2**	**0.998931**
7	3	0.998914
7	5	0.998911
7	7	0.998888
7	9	0.998918
8	1	0.998881
8	2	0.998864
8	3	0.998858
9	1	0.998841
9	3	0.998849
9	5	0.998858
9	7	0.998784
9	9	0.998865

(b)

`eta`	=	0.08
`gamma`	=	0
`subsample`	=	1
`colsample_bytree`	=	1
`lambda`	=	1

section 3.2) are shown in table 3.5a, while the last two rounds (d and e in section 3.2) are shown in 3.5b. The values of the other already tuned parameters that are kept constant are shown in 3.5c.

The optimal value of `lambda` resulting from the tuning is 46 improving the performance to AUC $= 0.998981$ (as shown in table 3.5). Since `lambda` is the variable that determines the amount of regularisation it has a large range of possible values. A larger number of tuning rounds were therefore performed.

The number of signal events is not equal to the number of background events and such unbalanced classes can influence the learning of the BDT. Therefore a rescaling of the weights that will balance the classes is tested to see if it improves the performance of

Table 3.3: Tuning results for `gamma`. In (a) the different values of `gamma` are shown with the AUC of the resulting BDT. The values in bold indicate the best values resulting from the tuning. In (b) the values of the other parameters that are kept constant during this tuning round are shown.

(a)

`gamma`	AUC
0.0	**0.998931**
0.1	0.998906
0.2	0.998913
0.3	0.998913
0.4	0.998921
0.5	0.998875

(b)

`eta`	=	0.08
`max_depth`	=	7
`min_child_weight`	=	2
`subsample`	=	1
`colsample_bytree`	=	1
`lambda`	=	1

the BDT where all the other parameters are already tuned. When scale is set to False, the parameter `scale_pos_weight` is set to 1 (the default value) and no rescaling is done. However, when scale is set to True, the parameter `scale_pos_weight` is set to the typical value of *(number of events of class 0)/(number of events of class 1)*, where class 0 corresponds to background and class 1 corresponds to signal. The results of comparing the performance of a BDT for these values of `scale_pos_weight` are shown in table 3.6.

The classes are not unbalanced enough for the rescaling of the weights by `scale_pos_weight` to improve performance (see table 3.6).

The resulting values for the parameters after the tuning procedure are summarised below. The BDT using these values has a AUC = 0.998981 for the test data.

- `eta` = 0.08
- `max_depth` = 7
- `min_child_weight` = 2
- `gamma` = 0
- `subsample` = 0.9
- `colsample_bytree` = 0.7
- `lambda` = 46
- `scale_pos_weight` = 1 (scale = False)

The AUC given is for the best iteration as found using `early_stopping_rounds` = 10. This best iteration corresponds to `num_round` = 197. The best model is the BDT trained with the variables resulting from the tuning procedure (including `num_round` = 197). To make sure that the number of boosting iterations selected by the early stopping procedure is accurate and no overfitting (or underfitting) occurs, the AUC for the test and train data

Table 3.4: Tuning results for `subsample` and `colsample_bytree`. In (a) and (b) the different values of `subsample` and `colsample_bytree` are shown with the AUC of the resulting BDT for respectively the first and second tuning round. The values in bold indicate the best values resulting from the tuning. In (c) the values of the other parameters that are kept constant during this tuning round are shown.

(a)

`subsample`	`colsample_bytree`	AUC
0.5	0.5	0.998862
0.5	0.6	0.998901
0.5	0.7	0.998902
0.5	0.8	0.998906
0.5	0.9	0.99892
0.5	1.0	0.998905
0.6	0.5	0.998905
0.6	0.6	0.998891
0.6	0.7	0.998929
0.6	0.8	0.998935
0.6	0.9	0.998921
0.6	1.0	0.998915
0.7	0.5	0.998892
0.7	0.6	0.998904
0.7	0.7	0.998928
0.7	0.8	0.998934
0.7	0.9	0.99889
0.7	1.0	0.998946
0.8	0.5	0.998858
0.8	0.6	0.998897
0.8	0.7	0.998909
0.8	0.8	0.998918
0.8	0.9	0.99891
0.8	1.0	0.998878
0.9	0.5	0.998801
0.9	0.6	0.998881
0.9	**0.7**	**0.998951**
0.9	0.8	0.998905
0.9	0.9	0.998924
0.9	1.0	0.998915
1.0	0.5	0.998872
1.0	0.6	0.998894
1.0	0.7	0.998919
1.0	0.8	0.99889
1.0	0.9	0.998876
1.0	1.0	0.998931

(b)

`subsample`	`colsample_bytree`	AUC
0.85	0.6	0.998901
0.85	0.7	0.998915
0.85	0.8	0.998917
0.95	0.6	0.998897
0.95	0.7	0.998885
0.95	0.8	0.998917

(c)

`eta`	=	0.08
`max_depth`	=	7
`min_child_weight`	=	2
`gamma`	=	0
`lambda`	=	1

of a BDT using all the tuned values for the parameters, but with `num_round` = 600 is shown in figure 3.4.

Table 3.5: Tuning results for `lambda`. In (a) and (b) the different values of `lambda` shown with the AUC of the resulting BDT for respectively the first three and last two tuning rounds. The values in bold indicate the best values resulting from the tuning. In (c) the values of the other parameters that are kept constant during this tuning round are shown.

(a)

`lambda`	AUC
0.0	0.998862
0.3	0.998875
0.7	0.99891
0.8	0.99894
0.9	0.998941
1.0	0.998951
1.1	0.998915
1.2	0.998919
1.3	0.998934
1.4	0.998923
1.5	0.998935
2	0.998923
5	0.998921
8	0.998947
10	0.998948
13	0.998951
14	0.99896
15	0.998962
16	0.998955
17	0.99894
18	0.998942
19	0.998959
20	0.998945
25	0.99895
30	0.998962
40	0.998976
50	0.998958
100	0.998965

(b)

`lambda`	AUC
32	0.99896
34	0.998962
36	0.998951
38	0.998978
39	0.998951
41	0.998971
42	0.99896
44	0.998977
45	0.998961
46	**0.998981**
47	0.998965
48	0.998977
500	0.998915
1000	0.998877

(c)

`eta`	=	0.08
`max_depth`	=	7
`min_child_weight`	=	2
`gamma`	=	0
`subsample`	=	0.9
`colsample_bytree`	=	0.7

The learning curve (see figure 3.4) shows that it is reasonable to stop training around `num_round` = 197, because at that point the AUC for the test data no longer increases and for a higher number of boosting iterations the AUC for the test data slowly starts to decrease. Note that this plot is similar to the illustration in figure 3.3, but mirrored on the y-axis in the sense that the AUC should be maximised, while the error in figure 3.3 should be minimised.

To get an unbiased estimate of the performance of the tuned BDT (the best model), it is tested on the independent test data set. The performance on the test data is biased, since it is used to determine the optimal value of the parameters in the tuning procedure, as well as the value for `num_round`.

Table 3.6: The AUC of the BDT with using rescaling of weights (scale = True setting `scale_pos_weight` = *(number of events of class 0)/(number of events of class 1)*) and without rescaling (scale = False setting `scale_pos_weight` = 1). The values in bold indicate the best values resulting from the tuning. In (b) the values of the other parameters that are kept constant during this tuning round are shown.

(a)

`scale`	AUC
False	**0.998981**
True	0.998954

(b)

`eta`	=	0.08
`max_depth`	=	7
`min_child_weight`	=	2
`gamma`	=	0
`subsample`	=	0.9
`colsample_bytree`	=	0.7
`lambda`	=	46

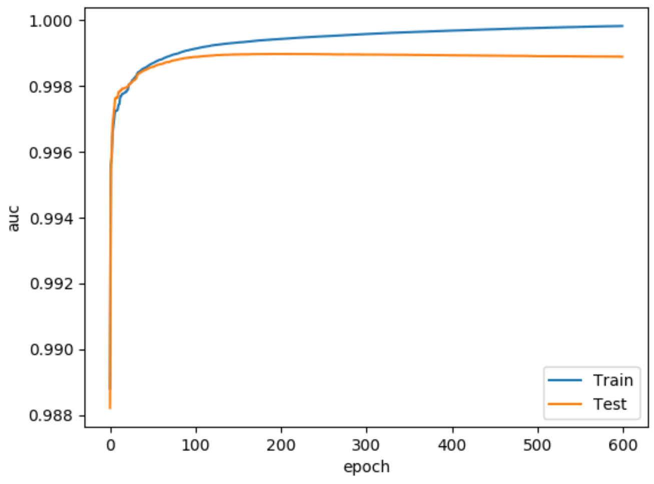


Figure 3.4: The AUC or the test and train data plotted as a function of the boosting iteration (also called epoch).

The resulting AUC value of the tuned BDT on the independent test set is given below.

- AUC = 0.998720

A number of other performance measures are also used to get a more complete image of the performance of the BDT on the independent test set. Those are all based on a threshold value of 0.5, where events with a score above 0.5 are classified as signal events and those with a score below 0.5 are classified as background events. Using this classification the accuracy gives the fraction of correctly classified events by dividing the sum of true positives and true negatives by the sum of all events. The precision is given by dividing the number of true positives by the sum of the true positives and false positives. The recall gives the True Positive Rate calculated by dividing the number of true positives by the sum of the true positives and false negatives. The True Negative Rate is given by the specificity which is calculated by dividing the number of true negatives by the sum of the true negatives and the false positives. The corresponding equations are given below for clarification,

$$
\begin{aligned}
\text{Accuracy} &= \frac{TP + TN}{TP + TN + FP + FN} \\
\text{Precision} &= \frac{TP}{TP + FP} \\
\text{Recall} &= \frac{TP}{TP + FN} \\
\text{Specificity} &= \frac{TN}{TN + FP}
\end{aligned}
$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives and FN is the number of false negatives. The values for these performance measures of the tuned BDT on the independent test set are given below.

- accuracy = 0.984280
- precision = 0.978814
- recall = 0.983764
- specificity = 0.984653

The ROC curve corresponding to the AUC for the BDT on the independent test set, which is independent of the choice of the threshold value, is shown in figure 3.5.

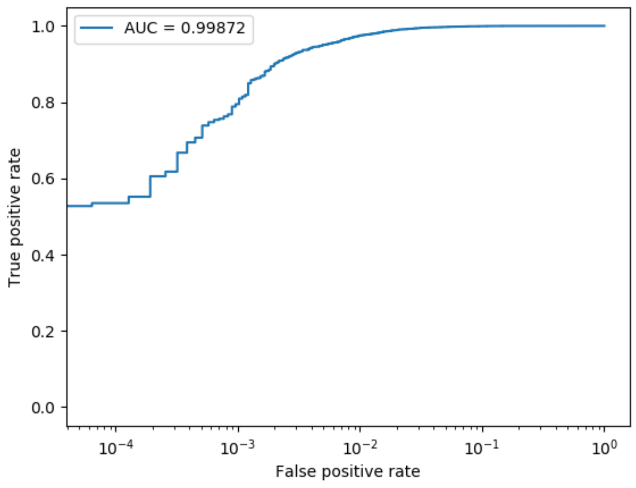


Figure 3.5: The ROC curve for the trained BDT on the independent test data on double logarithmic scale. The corresponding AUC value is also shown.

The ROC curve shows the True Positive Rate (the recall) as a function of the False Positive Rate. The False Positive Rate is calculated by dividing the number of false positives by the sum of false positives and true negatives, which is equal to 1 minus the specificity (True Negative Rate). Each point on the curve corresponds to the True Positive Rate and False Positive Rate for a specific threshold, which is varied to create the curve.

The distribution of the output score of the BDT, the probability that the event is a signal event, for the independent test data is shown for the signal and background events in figure 3.6.

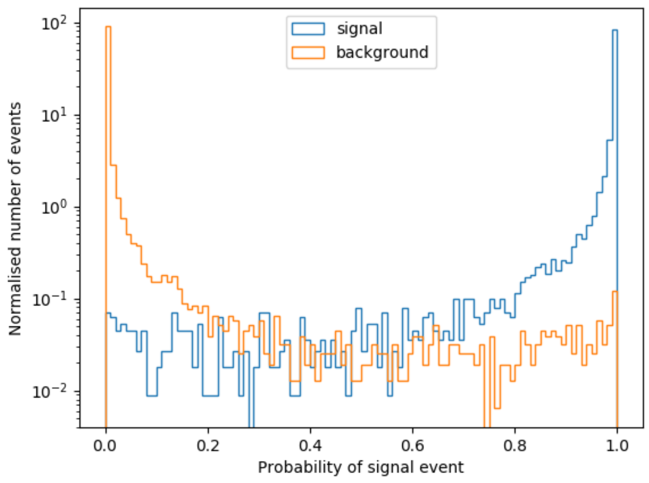


Figure 3.6: A histogram of the normalised number of signal and background events as a function of the probability that the event is a signal event as predicted by the tuned BDT for the independent test data.

Most events are correctly classified with a BDT score close to 0 or 1 for background and signal events respectively (see figure 3.6). However, a number of events are misclassified and can have value up to 1 for background events and down to 0 for signal events. The threshold used to convert the probability to the two different classes therefore also determines the trade-off between background rejection and signal efficiency. A high background rejection is preferred over a high signal efficiency, since any signal that would possibly be detected will have to be confirmed with a large certainty. For a background rejection of 99% (corresponding to a False Positive Rate of 0.01) the signal efficiency is 97% (corresponding to a True Positive Rate of 0.97) with a threshold of 0.70.

The BDT previously trained on a mixture of 1-electron events for 4 different mediator masses (0.001 GeV, 0.01 GeV, 0.1 GeV and 1 GeV) in [14], had signal efficiencies of 60% to 80% for the different mediator masses (about 77% for 0.01 GeV) for a background rejection of 99% with a threshold of 0.94. The performance of the BDT trained on 2-electron data (in this thesis) appears to be much better. However, there were some unphysical events present in the 1-electron data used to make that performance estimate. These type of events are no longer present in the 2-electron samples due to a modification of GEANT4. The performance of the BDT from [14] is also given for events where these unphysical events have been removed, which is a better performance estimate for comparison. For the same threshold of 0.94 the signal efficiency remains between 60% and 80% for the different mediator masses (about 77% for 0.01 GeV), but the False Positive Rate is reduced to $3 \cdot 10^{-4}$ (99.97% background rejection) [14]. The BDT trained here on 2-electron data has a signal efficiency of 62% and a threshold of 0.998 for a False Positive Rate of $3 \cdot 10^{-4}$. This is a lower signal efficiency and a higher threshold than the 1-electron BDT for the same background rejection. For a threshold of 0.94 the 2-electron BDT has a signal efficiency of 93% and a False Positive Rate of $3.2 \cdot 10^{-3}$. The performance of the 1-electron BDT seems to be somewhat better although there are some differences between the BDTs even though the same features are used. The 1-electron BDT was trained on a mixture of events for different mediator masses, while the 2-electron BDT was only trained on events for a mediator mass of 0.01 GeV, which makes comparing them a little harder.

This difference in performance could (partially) be due to the difference in signal and background separation of the features for 1-electron and 2-electron samples (as discussed in section 2.3), since the features were originally selected for the training of a BDT on 1-electron samples. The importance of the features in the training of the BDT on the 2-electron events is shown in figure 3.7.

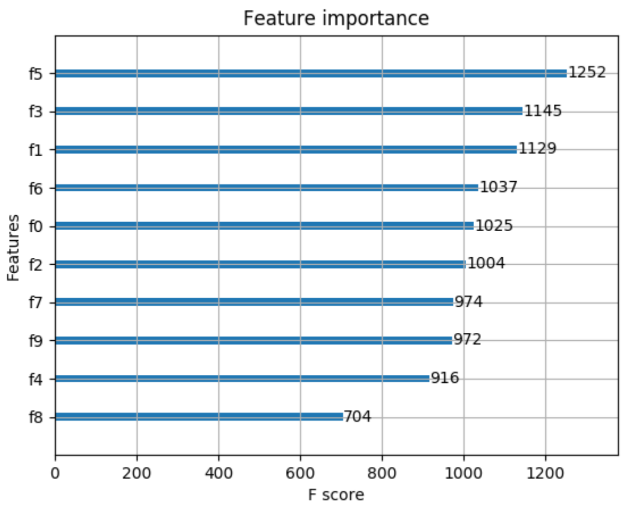


Figure 3.7: The feature importance in the BDT for all the 10 features used. See text for the features corresponding to the feature numbers.

The features with their corresponding number are shown below in order of importance (most important first, see figure 3.7).

- f5: Standard deviation of x position
- f3: Highest energy in a single cell
- f1: Total energy deposited
- f6: Standard deviation of y position
- f0: Number of readout hits
- f2: Total tight isolated energy deposited
- f7: Average layer hit

  - f9: Standard deviation of layers hit
  - f4: Transverse RMS
  - f8: Deepest layer hit

The importance of a feature (called the F score) is based on how often the feature is used to make a split and how much it improved the performance of signal and background separation. The distributions of these features are shown in section 2.3. The difference in the importance in not that large for most features, but there are some notable aspects of the ranking. The highest energy in a single cell seemed to give a poor separation between signal and background (see figure 2.9a), but has the second largest importance. The transverse RMS appears to have a decent separation of signal (with a mediator mass of 0.01 GeV) and background events (see figure 2.13a), but is the second lowest importance. A possible explanation for this is that events that are separated by the transverse RMS have already been separated using other features, such as the standard deviation of the x and y positions, since these features are related, making the feature transverse RMS less important. The highest energy in a single cell possibly separates some events that are not really separated by the other features, making it more important. The other features that appeared to give a good separation, from looking at their distribution in section 2.3, are all in the top part of the importance ranking, while those that appeared to give a poor separation are in the bottom part of the importance ranking.

Another reason for the decrease in performance of the 2-electron BDT compared to the 1-electron BDT is that the second electron in the 2-electron samples does not add any discrimination power, but only makes the differentiation between signal and background more difficult. However, having two electrons simultaneously hitting the target also has advantages. The biggest advantage of the 2-electron case is that it has twice as many electrons on target, which means that probability of a dark matter event is much larger within the same amount of measuring time compared to the 1-electron case. This advantage outweighs the slight reduction in performance of the BDT.

4 | Conclusion

The energy and transverse momentum distributions are shown for 2-electron signal simulation samples for a mediator mass of 0.01 GeV, 0.1 GeV and 1 GeV as well as for the PN background simulation samples. For these samples the distributions of the features used to train the BDT are also shown and compared to the distributions for the 1-electron signal and background samples. For most of the features the separation between the distributions for the signal and background samples is fairly similar between the 1-electron and 2-electron samples. However, for the average layer hit and the standard deviation of layer hits the separation between distributions for signal and background samples is a lot smaller for the 2-electron samples compared to the 1-electron samples, due to the effect of the energy deposit of the second electron.

A BDT was trained on the 2-electron signal simulation sample for a mediator mass of 10 MeV (0.01 GeV) and a 2-electron PN background sample. After tuning a number of parameters the BDT has a performance of AUC = 0.998981 on the test data. The unbiased measure of the performance of the BDT on the independent test set is AUC = 0.998720. The BDT gives a signal efficiency of 97% with a threshold of 0.7 for a background rejection of 99% (corresponding to a False Positive Rate of 0.01). The performance of the BDT trained on 1-electron data in [14] is slightly better than the 2-electron trained in this thesis, because the 1-electron BDT has a signal efficiency between 60% and 80% for the different mediator masses (about 77% for 0.01 GeV) and a threshold of 0.94 for a False Positive Rate of $3 \cdot 10^4$ (99.97% background rejection), while the 2-electron BDT has a signal efficiency of 62% and a threshold of 0.998 for the same background rejection. However, the 1-electron BDT was trained on a mixture of events for different mediator masses, while the 2-electron BDT is trained solely on events for a mediator mass of 0.01 GeV. This difference in methods makes comparing them more difficult. One of the possible reasons for the difference in performance of the 1-electron and 2-electron BDTs is that the features that are used for both were originally selected based on the 1-electron samples. Another reason for the decrease in performance is that the second electron adds no the discrimination power, but only complicates the differentiation between signal and background events. However, the 2-electron case has twice as many electrons on target, which is a big advantage that outweighs the slight decrease of performance for the BDT.

5 | Outlook

As shown in section 2.3 (figure 2.7a), the sum of energy deposited in the ECal is larger than 8 GeV, which is the maximum energy that could be deposited (from two 4 GeV beam electrons). This motivates the need for a better energy calibration of the ECal.

To improve the performance of the BDT other features, more specialised for 2-electron data, should be explored. Especially features which are not calculated using the whole ECal, but are more shower-specific, since 2-electron events have multiple showers in the ECal. Ideally, features that are good discriminators for all multi-electron events (n electrons) should be found. A BDT could also be trained on signal simulations samples for different mediator masses or on a combination of events for different mediator masses to make it less dependent on the hypothesis of the mediator mass.

Acknowledgements

First of all I would like to thank my supervisor Prof. Torsten Åkesson for making this thesis possible. I would like to thank my daily supervisor Dr. Ruth Pöttgen for the great guidance she has provided during the project. I would also like to thank the Lund LDMX group for the useful discussion and suggestions and the entire LDMX collaboration for their input. Lastly I would like to thank everyone from the particle physics department for making me feel at home with (among other things) the weekly fika.

Acronyms

AUC Area Under Curve.

BDT Boosted Decision Tree.

ECal Electromagnetic Calorimeter.

HCal Hadron Veto Calorimeter.

LDMX Light Dark Matter Experiment.

PN photo-nuclear.

RMS Root Mean Square.

RMSE Root Mean Square Error.

ROC Receiver Operating Characteristic.

SM Standard Model.

WIMPs Weakly Interacting Massive Particles.

A | Python script BDT

```python
#!/usr/bin/env python

import numpy as np
import xgboost as xgb
import ROOT as r
import os
import sys
import time
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import argparse
import matplotlib
matplotlib.use('Agg') #needed when running with bsub
import matplotlib.pyplot as plt


#---- From bdtTreeMaker.py -----
cellMap = np.loadtxt('cellmodule.txt')
ecalFaceZ = 223.8000030517578
cell_radius = 5

def CallX(Hitz, Recoilx, Recoily, Recoilz, RPx, RPy, RPz):
    Point_xz = [Recoilx, Recoilz]
    #Almost never happens
    if RPx == 0:
        slope_xz = 99999
    else:
        slope_xz = RPz / RPx

    x_val = (float(Hitz - Point_xz[1]) / float(slope_xz)) + Point_xz[0]
    return x_val

def CallY(Hitz, Recoilx, Recoily, Recoilz, RPx, RPy, RPz):
    Point_yz = [Recoily, Recoilz]
    #Almost never happens
    if RPy == 0:
        slope_yz = 99999
    else:
        slope_yz = RPz / RPy

    y_val = (float(Hitz - Point_yz[1]) / float(slope_yz)) + Point_yz[0]
    return y_val
#--------------------------------

# Both electrons inside fiducial region or not
def InFiducialTwo(event):
    insideEcal = False

    mom = []
    pos = []

    #find momentum of largest hit of recoil e in first ecal scoring plane
    for particle in event.SimParticles_sim:
        if (particle.getPdgID()==11) & (particle.getParentCount()==0):
            momentum = []
            position = []
            max_p = 0. #for finding the hit with largest momentum
            for hit in event.EcalScoringPlaneHits_sim:
                if hit.getSimParticle() == particle:
                    #only look at front plane Ecal
                    if hit.getPosition()[2] > 219.85 and hit.getPosition()[2] < 220.05:
                        pvec = r.TVector3(hit.getMomentum()[0],hit.getMomentum()[1],hit.getMomentum()[2])
                        p = pvec.Mag()
                        if max_p < p and hit.getMomentum()[2] > 0:
                            max_p = p
                            momentum = [hit.getMomentum()[0],hit.getMomentum()[1],hit.getMomentum()[2]]
                            position = [hit.getPosition()[0],hit.getPosition()[1],hit.getPosition()[2]]
            mom.append(momentum)
            pos.append(position)

    insideBoth = []

    #see if recoil e inside fiducial region, for both e's
    for i in range(len(mom)):
        insideSingle = False

        #only for events where a hit with positive z momentum is found
        if len(mom[i]) > 0:
            #use real z position hit, not scoringPlaneZ = 220
            recoilfX = CallX(ecalFaceZ, pos[i][0], pos[i][1], pos[i][2], mom[i][0], mom[i][1], mom[i][2])
            recoilfY = CallY(ecalFaceZ, pos[i][0], pos[i][1], pos[i][2], mom[i][0], mom[i][1], mom[i][2])

            if not pos[i][0] == -9999 and not pos[i][1] == -9999 and not
            ↪ mom[i][0] == -9999 and not mom[i][1] == -9999 and not
            ↪ mom[i][2] == -9999:
                for x in cellMap:
                    xdis = recoilfY - x[2]
                    ydis = recoilfX - x[1]
                    celldis = np.sqrt(xdis**2 + ydis**2)
                    if celldis <= cell_radius:
                        insideSingle = True
                        break

        insideBoth.append(insideSingle)

    #return true if both e's inside the Ecal
    if insideBoth[0] and insideBoth[1]:
        insideEcal = True

    return insideEcal


#####################################################################
class prepSample:
    def __init__(self, fn, trainFrac, isSig):
        #bool to distinguish bkg file from sig file
        self.isSig = isSig

        self.fn = fn
        self.trainFrac = trainFrac

        #to store events and their relevant features
        self.events = []

    def createEventsArray(self):
        #get tree from file
        f = r.TFile(self.fn)
        tree = f.Get("LDMX_Events")

        #loop through events to same relevant variables
        for event in tree:
            #only look at events where both e's are within fiducial
            ↪ region
            if not InFiducialTwo(event): continue

            temp_evt = []

            #------------- FEATURES (same as slac example)--------------
            temp_evt.append(event.EcalVeto_recon[0].getNReadoutHits())
            temp_evt.append(event.EcalVeto_recon[0].getSummedDet())
            temp_evt.append(event.EcalVeto_recon[0].getSummedTightIso())
            temp_evt.append(event.EcalVeto_recon[0].getMaxCellDep())
            temp_evt.append(event.EcalVeto_recon[0].getShowerRMS())
            temp_evt.append(event.EcalVeto_recon[0].getXStd())
            temp_evt.append(event.EcalVeto_recon[0].getYStd())
            temp_evt.append(event.EcalVeto_recon[0].getAvgLayerHit())
            temp_evt.append(event.EcalVeto_recon[0].getDeepestLayerHit())
            temp_evt.append(event.EcalVeto_recon[0].getStdLayerHit())
            #----------------------------------------------------------

            self.events.append(temp_evt)

        #convert list of lists to nd numpy array
        self.events = np.array(self.events)

        #shuffle order (only in 0th, vertical axis)
        self.events = np.random.permutation(self.events)

        print "shape of data array:", np.shape(self.events)

    def createTrainTestArrays(self):
        #only use 80% of data to have 20% of data left for independent test
        data_using = self.events[0:int(len(self.events)*0.8)]
        self.indep_test_x = self.events[int(len(self.events)*0.8):]

        self.train_x = data_using[0:int(len(data_using)*self.trainFrac)]
        self.test_x  = data_using[int(len(data_using)*self.trainFrac):]

        self.train_y = np.zeros(len(self.train_x)) + self.isSig
        self.test_y  = np.zeros(len(self.test_x)) + self.isSig

#######################################################################

class mergeSigBkg:
    def __init__(self, sig, bkg):
        #create a training set with both sig and background
        self.train_x = np.vstack((sig.train_x, bkg.train_x))
        self.train_y = np.append(sig.train_y, bkg.train_y)

        #create a testing set with both sig and background
        self.test_x  = np.vstack((sig.test_x, bkg.test_x))
        self.test_y  = np.append(sig.test_y, bkg.test_y)

        print "shape training data, x and y (sig+bkg):", np.shape(self.train_x), np.shape(self.train_y)
        print "shape testing data, x and y (sig+bkg):", np.shape(self.test_x), np.shape(self.test_y)

        #create xgb Dmatrices of data
        self.dtrain = xgb.DMatrix(self.train_x, self.train_y)
        self.dtest = xgb.DMatrix(self.test_x, self.test_y)

#######################################################################

```

```
def getDataBDT(trainFraction):
    #prepare data signal sample
    sig = prepSample(sigFile, trainFraction, isSig=True)
    sig.createEventsArray()
    sig.createTrainTestArrays()

    #prepare data bkg sample
    bkg = prepSample(bkgFile, trainFraction, isSig=False)
    bkg.createEventsArray()
    bkg.createTrainTestArrays()

    #merge sig and bkg events
    events = mergeSigBkg(sig, bkg)

    return events

def train(events, params, num_round, stopping_rounds, name_model):
    # to watch performance during learning
    evallist  = [(events.dtrain,'train'), (events.dtest,'eval')]
    learning = dict()

    #train bdt
    bst = xgb.train(params, events.dtrain, num_round, evallist,
       ↪ early_stopping_rounds=stopping_rounds, evals_result=learning)

    try:
        #find out if early stopping occured
        if bst.best_iteration != (num_round -1):
            early_stop = True
        else:
            early_stop = False
    except:
        early_stop = False

    #save model to file
    bst.save_model(os.path.join(folder,"%s.model" %name_model)) #model
    bst.dump_model(os.path.join(folder,"dump.raw.txt")) #structure of
       ↪ trees

    if early_stop:
        return bst, learning, early_stop, bst.best_score, bst.
           ↪ best_iteration
    else:
        return bst, learning, early_stop, np.nan, np.nan

def predict(bst, events):
    #predict the test values (gives probability for class 1)
    preds = bst.predict(events.dtest)

    #get classes from probability
    preds_bool = preds > 0.5 #0.5 cutoff, decision boundary
```

```python
    classes = preds_bool.astype(int)

    precision = precision_score(events.test_y, classes)
    print "prediction precision:", precision
    accuracy = accuracy_score(events.test_y, classes)
    print "prediction accuracy:", accuracy
    recall = recall_score(events.test_y, classes)
    print "prediction recall:", recall

    tn, fp, fn, tp = confusion_matrix(events.test_y, classes).ravel()
    specificity = float(tn) / (tn+fp)
    print "prediction specificity:", specificity

    #separate probability predictions of sig and bkg events
    probs_sig_events = preds[np.where(events.test_y==1)]
    probs_bkg_events = preds[np.where(events.test_y==0)]

    return preds, probs_sig_events, probs_bkg_events, (precision, accuracy, recall, specificity)


def plot_probs(probs_sig, probs_bkg):
    #plot predicted probalities
    binwidth = 0.005
    binrange = np.arange(0, 1 + binwidth, binwidth)

    np.savetxt(os.path.join(folder,"probability_hist_sig.csv"), np.transpose(probs_sig),delimiter=",", header="probability signal, probability bkg")
    np.savetxt(os.path.join(folder,"probability_hist_bkg.csv"), np.transpose(probs_bkg),delimiter=",", header="probability signal, probability bkg")

    plt.hist(probs_sig, bins=binrange, histtype='step', label='signal', density=True)
    plt.hist(probs_bkg, bins=binrange, histtype='step', label='background', density=True)
    plt.ylabel("Normalised number of events")
    plt.xlabel("Probability of signal event")
    plt.yscale('log')
    plt.legend(loc='best')
    plt.savefig(os.path.join(folder, "probability_hist.png"),format='png')
    plt.close()

def plot_roc(test_y, preds):
    #plot ROC curve
    fpr, tpr, thresholds = metrics.roc_curve(test_y, preds)
    auc = metrics.auc(fpr, tpr)

    np.savetxt(os.path.join(folder,"ROC_curve.csv"), np.transpose([fpr,
    ↪ tpr, thresholds]), delimiter=",", header="false positive rate,
    ↪ true positive rate, thresholds")

    plt.plot(fpr, tpr, label="AUC = %0.5f" %auc)
    plt.xlabel("False positive rate")
    plt.ylabel("True positive rate")
    plt.xscale('log')
    plt.legend(loc='best')
    plt.savefig(os.path.join(folder, "ROC_curve.png"),format='png')
    plt.close()

    return auc

def plot_learning(results, error_type):
    x = np.arange(len(results['train'][error_type]))

    np.savetxt(os.path.join(folder,"learning_curve_%s.csv" %error_type),
       ↪ np.transpose([x, results['train'][error_type], results['eval'][
       ↪ error_type]]), delimiter=",", header="epoch, training error,
       ↪ test error \n error metric: %s" %error_type)

    plt.plot(x, results['train'][error_type], label="Train")
    plt.plot(x, results['eval'][error_type], label="Test")
    plt.xlabel("epoch")
    plt.ylabel(error_type)
    plt.legend(loc='best')
    plt.savefig(os.path.join(folder, "learning_curve_%s.png" %error_type)
       ↪ ,format='png')
    plt.close()

####################################################################

def main(args, sigFile, bkgFile):

    print "Executing main() in BDT1.py"
    #-------------------------- BDT VARIABLES ------------------------
    trainFraction = 0.8
    seed = 2

    params = {"objective": "binary:logistic",
                "eta": args.eta,
                "max_depth": args.max_depth,
                "min_child_weight": args.min_child_weight,
                "gamma": args.gamma,
                "subsample": args.subsample,
                "colsample_bytree": args.colsample_bytree,
                "lambda": args.reg_lambda,
                "eval_metric": ['rmse', 'error', 'logloss', 'auc'],
                "seed": 5,
                "silent": 1,
                "verbosity": 1}
```

```python

    num_round = args.num_round #number of training iterations (determines number of boosters)
    early_stopping_rounds = args.early_stopping_rounds #make None to deactivate early stopping
    #--------------------------------------------------------------

    #to get identical result with same data and variables
    np.random.seed(seed)

    #----- BDT -----
    events = getDataBDT(trainFraction)

    if args.scale:
        params["scale_pos_weight"] = float(np.sum(events.train_y == 0))/np.sum(events.train_y == 1)
        print "Using scale_pos_weight"

    bst, learning, early_stop, best_score, best_iteration = train(events, params, num_round, early_stopping_rounds, args.name_model)
    preds, probs_sig_events, probs_bkg_events, scores = predict(bst, events)

    #----- plotting ------
    print "plotting"

    plot_probs(probs_sig_events, probs_bkg_events)
    auc = plot_roc(events.test_y, preds)

    for metric in params['eval_metric']:
        plot_learning(learning, metric)

    #plot importance
    ax = xgb.plot_importance(bst)
    fig = ax.figure
    fig.savefig(os.path.join(folder,"importance.png"))

    #----- save parameters used ------
    with open(os.path.join(folder,"variables.txt") ,"w") as f:
        f.write('CONSTANTS' + '\n\n')
        f.write('training fraction = ' + str(trainFraction) + '\n')
        f.write('seed = ' + str(seed) + '\n\n')
        f.write('PARAMETERS BDT' + '\n\n')
        for k, v in sorted(params.iteritems()):
            f.write(str(k) + ': ' + str(v) + '\n')
        f.write('\n' + 'number of rounds = ' + str(num_round) + '\n')
        f.write('early stopping rounds = ' + str(early_stopping_rounds) + '\n\n')
        f.write('RESULTS' + '\n\n')
        f.write('auc = ' + str(auc) + '\n')
        f.write('precision = ' + str(scores[0]) + '\n')
        f.write('accuracy = ' + str(scores[1]) + '\n')
        f.write('recall = ' + str(scores[2]) + '\n')
        f.write('specificity = ' + str(scores[3]) + '\n')
        f.write('metric values of last booster (test):' + '\n')
        for metric in params['eval_metric']:
            f.write('   - ' + str(metric) + ': ' + str(learning['eval'][
               ↪ metric][-1]) + '\n')
        f.write('\n')
        if early_stop:
            f.write('EARLY STOPPING OCCURED \n')
            f.write('best score = ' + str(best_score) + '\n')
            f.write('best iteration = ' + str(best_iteration) + '\n')
            f.write('metric values of best booster (test):' + '\n')
            for metric in params['eval_metric']:
                f.write('   - ' + str(metric) + ': ' + str(learning['eval
                   ↪ '][metric][best_iteration]) + '\n')
            f.write('\n')
        else:
            f.write('early stop = ' + str(early_stop) + '\n\n')
        f.write('INPUT FILES' + '\n\n')
        f.write('signal sample file: ' + sigFile + '\n')
        f.write('bkg sample file: ' + bkgFile)
    #--------------------------------
    print "Completed main() in BDT1.py"


if __name__ == "__main__":
    start_time = time.time()

    #load dictionaries (library)
    r.gSystem.Load("/nfs/slac/g/ldmx/users/jessamy/ldmx_git/ldmx-sw/
       ↪ install/lib/libEvent.so")

    #input files
    sigFile = "../plotting_2e/Data_2e/4
       ↪ pt0_gev_signal_ap_mass_10mev_recon_recon.root"
    bkgFile = "../plotting_2e/Data_2e/4
       ↪ pt0_gev_2e_ecal_pn_v9_magnet_20180825_1e6_fcbb5f56_tskim_recon_
       ↪  merge_recon.root"

    #give variables when running code
    parser = argparse.ArgumentParser()

    parser.add_argument('--eta', dest='eta',type=float,  default=0.3,
       ↪ help='Learning Rate')
    parser.add_argument('--max_depth', dest='max_depth',type=int,
       ↪ default=6, help='Maximum depth of trees')
    parser.add_argument('--min_child_weight', dest='min_child_weight',
       ↪ type=int,  default=1, help='Minimum child weight')
    parser.add_argument('--gamma', dest='gamma',type=float,  default=0,
       ↪ help='Minimum loss reduction needed for split')
```

```python
    parser.add_argument('--subsample', dest='subsample',type=float, default=1, help='Fraction of events used per tree')
    parser.add_argument('--colsample_bytree', dest='colsample_bytree', type=float,  default=1, help='Fraction of features used per tree')
    parser.add_argument('--lambda', dest='reg_lambda',type=float, default=1, help='L2 regularization factor')
    parser.add_argument('--num_round', dest='num_round',type=int, default=10, help='Number of booster rounds/number of trees')
    parser.add_argument('--early_stopping_rounds', dest='early_stopping_rounds',type=int,  default=None, help='Number of early stopping rounds')
    parser.add_argument('--folder', dest='folder', default='Output_bdt', help='Name of output folder')
    parser.add_argument('--subfolder', dest='subfolder', default=None, help='Name of output subfolder')
    parser.add_argument('--name_model', dest='name_model', default='0001', help='Name of model')
    parser.add_argument('--scale', dest='scale', type=bool, default=False, help='Bool for using scale_pos_weights')

    args = parser.parse_args()

    print args.scale

    #create output directory (if not already excisting)
    if args.subfolder:
        if not os.path.exists(os.path.join(args.folder, args.subfolder)):
            os.makedirs(os.path.join(args.folder, args.subfolder))
        folder = os.path.join(args.folder, args.subfolder)
    else:
        if not os.path.exists(args.folder):
            os.mkdir(args.folder)
        folder = args.folder


    main(args, sigFile, bkgFile)


    #calc how long program took
    time_taken = time.time() - start_time #in seconds
    hours, rest = divmod(time_taken, 3600)
    minutes, seconds = divmod(rest, 60)

    print hours, "hours,", minutes, "minutes,", seconds, "seconds"
```

B | Python submission script

```python
#!/usr/bin/env python

import subprocess
import time
import os

def main():

    print "Running submission.py"

    command = ['bash', '-c', 'source /nfs/slac/g/ldmx/software/setup.sh
        ↪ && env']
    proc = subprocess.Popen(command, stdout=subprocess.PIPE)

    dir = "/nfs/slac/g/ldmx/users/jessamy/Thesis/SLAC/firstBDT/"

    folder = "tune_gamma"
    gamma_list = ['0.0','0.1','0.2','0.3','0.4','0.5']

    output_dir = dir + folder + '/'

    for gamma in gamma_list:
        sub_name = "gamma_%s" %(gamma)

        command = 'python BDT1.py --eta 0.08 --max_depth 7 --
            ↪ min_child_weight 2 --gamma %s --subsample 1 --
            ↪ colsample_bytree 1 --lambda 1 --num_round 1000 --
            ↪ early_stopping_rounds 10 --folder %s --subfolder %s --
            ↪ name_model %s' %(gamma, folder, sub_name, sub_name)

        print '\n', command, '\n'

        batch_command = "bsub -q medium -W 2800 -o %s -e %s %s" %(
            ↪ output_dir, output_dir, command)
        process = subprocess.Popen(batch_command, shell=True)
        process.wait()

    print "Finished submission.py"

if __name__ == "__main__" :
    main()
```

Bibliography

[1] F Zwicky. Die Rotverschiebung von extragalaktischen Nebeln. *Helvetica Physica Acta*, 6:110–127, 1933.

[2] Knut Lundmark. *Lund Medd.*, 125:1–10, 1930.

[3] NASA. Dark energy, dark matter. `https://science.nasa.gov/astrophysics/focus-areas/what-is-dark-energy`. Accessed: 2018-02-24.

[4] CERN. Dark matter. `https://home.cern/about/physics/dark-matter`. Accessed: 2018-02-24.

[5] `https://commons.wikimedia.org/wiki/File:Standard_Model_From_Fermi_Lab.jpg`. Accessed: 2018-02-24.

[6] V. C. Rubin, W. K. Ford, Jr., and N. Thonnard. Rotational properties of 21 SC galaxies with a large range of luminosities and radii, from NGC 4605 /R = 4kpc/ to UGC 2885 /R = 122 kpc/. *apj*, 238:471–487, June 1980.

[7] `https://en.wikipedia.org/wiki/File:GalacticRotation2.svg`. Accessed: 2018-02-25.

[8] `https://commons.wikimedia.org/wiki/File:A_Horseshoe_Einstein_Ring_from_Hubble.JPG`. Accessed: 2018-02-25.

[9] `https://commons.wikimedia.org/wiki/File:Gravitational_lens-full.jpg`. Accessed: 2018-02-25.

[10] Douglas Clowe, Anthony Gonzalez, and Maxim Markevitch. Weak-lensing mass reconstruction of the interacting cluster 1e 0657âĂŞ558: Direct evidence for the existence of dark matter. *The Astrophysical Journal*, 604(2):596, 2004.

[11] Douglas Clowe, MaruÅąa BradaÄŊ, Anthony H. Gonzalez, Maxim Markevitch, Scott W. Randall, Christine Jones, and Dennis Zaritsky. A direct empirical proof of the existence of dark matter. *The Astrophysical Journal Letters*, 648(2):L109, 2006.

[12] Planck Collaboration: Ade, P. A. R. et al. Planck 2015 results - xiii. cosmological parameters. *A&A*, 594:A13, 2016.

[13] Howard Baer, Ki-Young Choi, Jihn E. Kim, and Leszek Roszkowski. Dark matter production in the early universe: Beyond the thermal WIMP paradigm. *Physics Reports*, 555:1 – 60, 2015.

[14] Torsten Åkesson, Asher Berlin, Nikita Blinov, Owen Colegrove, Giulia Collura, Valentina Dutta, Bertrand Echenard, Joshua Hiltbrand, David G. Hitlin, Joseph Incandela, John Jaros, Robert Johnson, Gordan Krnjaic, Jeremiah Mans, Takashi Maruyama, Jeremy McCormick, Omar Moreno, Timothy Nelson, Gavin Niendorf, Reese Petersen, Ruth Pöttgen, Philip Schuster, Natalia Toro, Nhan Tran, and Andrew Whitbeck. Light Dark Matter eXperiment (LDMX). *arXiv e-prints*, page arXiv:1808.05219, Aug 2018.

[15] J. et al. Alexander. Dark Sectors 2016 Workshop: Community Report. *ArXiv e-prints*, August 2016.

[16] Teresa MarrodÃąn Undagoitia and Ludwig Rauch. Dark matter direct-detection experiments. *Journal of Physics G: Nuclear and Particle Physics*, 43(1):013001, 2016.

[17] Mans, Jeremiah. The LDMX Experiment. *EPJ Web Conf.*, 142:1020, 2017.

[18] S. Agostinelli et al. GEANT4: A Simulation toolkit. *Nucl. Instrum. Meth.*, A506:250–303, 2003.

[19] Dan Guest, Kyle Cranmer, and Daniel Whiteson. Deep learning and its application to lhc physics. *Annual Review of Nuclear and Particle Science*, 68(1):161–181, 2018.

[20] Xgboost documentation. `https://xgboost.readthedocs.io/en/latest/index.html`. Accessed: 2019-04-02.

[21] Byron P. Roe, Hai-Jun Yang, Ji Zhu, Yong Liu, Ion Stancu, and Gordon McGregor. Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 543(2):577 – 584, 2005.

[22] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system, 2016.

[23] XGBoost documentation. Introduction to boosted trees. `https://xgboost.readthedocs.io/en/latest/tutorials/model.html`. Accessed: 2019-04-01.

[24] Tianqi Chen. Introduction to boosted trees. `https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf`, 2014. Accessed: 2019-04-01.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[26] Julien Despois. Memorizing is not learning! -6 tricks to prevent overfitting in machine learning. `https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42`. Accessed: 2019-04-01.

[27] XGBoost Documentation. Xgboost parameters. `https://xgboost.readthedocs.io/en/latest/parameter.html`. Accessed: 2019-04-02.

[28] XGBoost Documentation. Python API reference. `https://xgboost.readthedocs.io/en/latest/python/python_api.html`. Accessed: 2019-04-02.