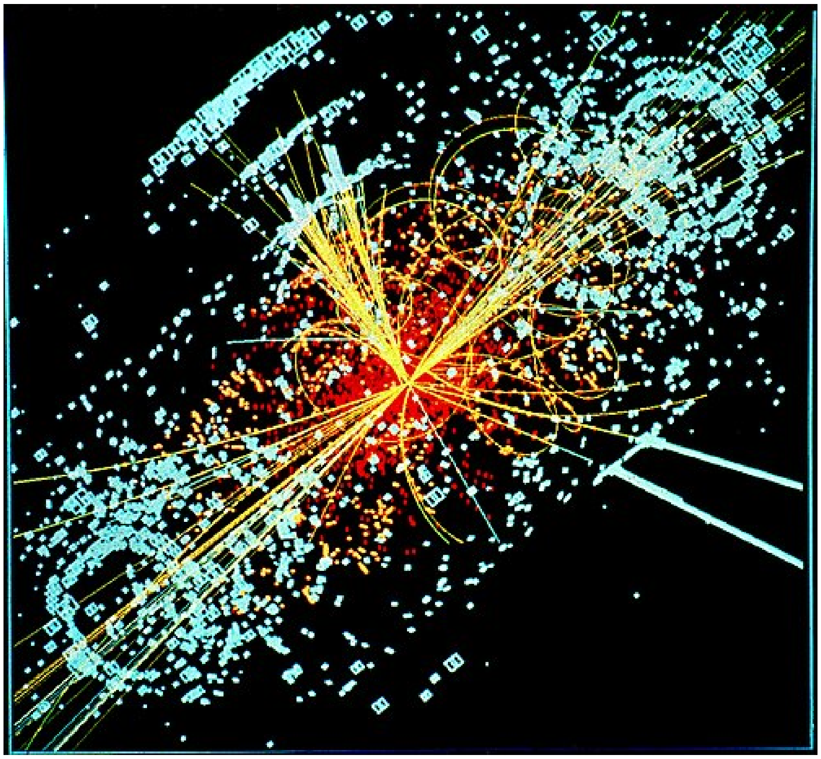
Advanced Machine Learning

Searching for the Higgs through supervised and unsupervised machine learning algorithms

Sakarias Frette & William Hirst

Spring 2022



Source, accessed 23.03.22

University of Oslo

CONTENTS

Abstract

An abstract is a self-contained summary of a larger work, such as research and scientific papers or general academic papers. Usually situated at the beginning of such works, the abstract is meant to preview the bigger document. This helps readers and other researchers find what theyre looking for and understand the magnitude of whats discussed.

I. INTRODUCTION

The Higgs boson was the "final" piece of the Standard Model puzzle, being experimentally proven at the LHC in 2012. Through spontaneous symmetry breaking it is responsible for giving mass to fermions and other bosons. It's discovery in 2012 showed a 5.9 sigma[1] significance, concluding with high certainty that they discovered the particle. Its discovery was shown through the decay of the Higgs boson into boson pairs, as well as decay to bottom and anti bottom quarks, and tau and anti tau leptons.

The ATLAS experiment is one of the largest particle detector experiments in the world. Alone it generates approximately 1 petabyte of raw data every second from proton proton collisions. With great developments in science the amount of data will only increase, therefore taking advantage of numerical methods like machine learning is pivotal if scientific development is to keep up with technological development. In this report we aim to compare the efficiency (accuracy and speed) of multiple machine learning methods and data analysis tools to discover Higgs bosons in ATLAS dataset, a competion that was run in 2014[1].

The report is structured in a theory, implementation, results/discussion and conclusion section. In the theory section we will introduce necessary theory, both with regards to the data and its respected contributions as well as the ML methods and our choice of evaluating them. In the implementation section we discuss data handling, the software of choice as well as grid-searching for optimal parameters for each method. In the results and discussion section we present all results as well as discuss any notible observations. In the final section, conclusion we summarize all notable results and observations of interest.

II. THEORY

A. Higgs boson decay

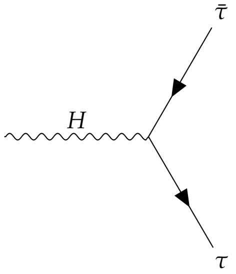


Figure 1: Feynman diagram for dilepton final state for the decay of Higgs into τ, $\bar{\tau}$.

The Higgs boson has a short mean lifespan of about $1.6 \cdot 10^{-22}$, which leaves no hope for any detector of measuring it. In fact, today's detectors can detect everything with a lifespan of down to $\approx 10^{-10}$s. This forces us to look at the possible decays of the Higgs boson, and measure those particles instead. This leads us to branching ratio. The branching ratio is a measurement the fraction of total decays for a given decay mode. The branching ratio of the Higgs boson decay is shown in the figure below.

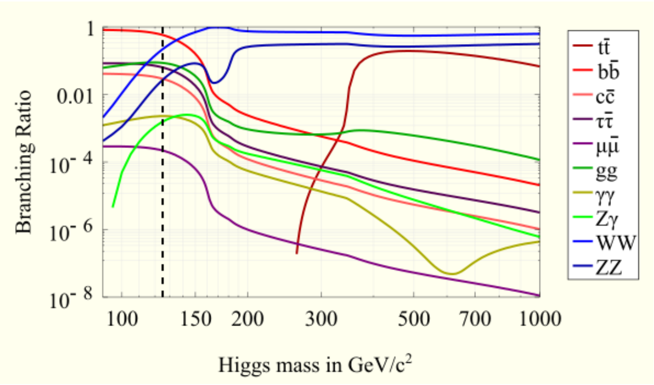


Figure 2: Branching ratio of Higgs decay as function of Higgs mass. Source, accessed 23.03.22

In figure 2 we see the branching ratio of the Higgs boson decay. Here we see that for energies lower and equal to the Higgs mass of 125.1 GeV the $H \rightarrow b\bar{b}$ decay is the most occurring decay. At higher energies we see that decay into vector bosons takes over, with only

[1] Competition was posted on Kaggle.com with over 1700 participants https://higgsml.ijclab.in2p3.fr/documentation/

$t\bar{t}$ as a significant fraction representing the decay into fermions.

This challenge aims to analyse the decay of $H \rightarrow \tau\bar{\tau}$. Its difficulty arrives from the fact that both the W- and the Z-boson can decay into the τ lepton. In the case of $W^- \rightarrow \tau\bar{\nu}_\tau$ we also get a neutrino in the final state which cannot be detected in the detector. This leads to a problem of evaluating the Higgs mass on a event by event basis. In the case of $Z \rightarrow \tau\bar{\tau}$ we have that the Higgs mass and Z mass are very close, 125 GeV and 91 GeV respectively. Due to the closeness of their mass, they are difficult to separate. There is also the problem that Z decays more often to $\tau\bar{\tau}$ than Higgs.

B. The background signal

The data which will be analysed in this paper was produced by Monte Carlo simulations and contains three, well known background processes and one signal. The first of the three is a Z-boson decay through two τ-leptons. This diagram is quite similar to the signal process and could therefore lead to problems when separating signal from background. The second diagram is the decay of a W-boson into a charged and neutral lepton, l, ν_l. The first order diagram of the Z- and W-decay is shown in figure 3.

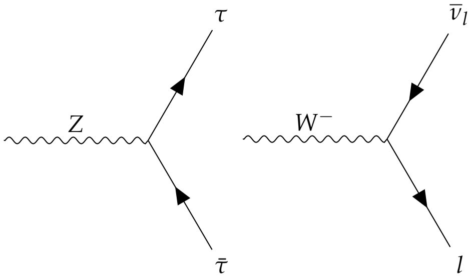


Figure 3: Feynman diagram for the Z-boson decay with two τ-leptons and W-boson decay to charged and neutral lepton.

The third and final background process is the hadronic decay of a pair of top-quark. The top-quark of the pair decays into a bottom-quark and a hadron mediated by the W boson. Half of the third process is shown in figure 4.

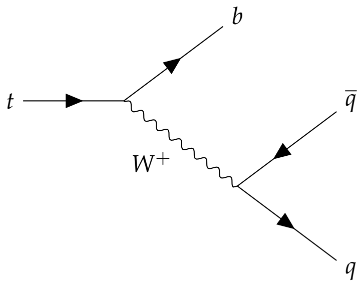


Figure 4: Feynman diagram for the top quark decay with 3 quark final state

C. Supervised Machine learning

The in depth theory of the different models will not be written here, however there will be a summary of the different methods. For an in depth theory of neural networks, we wrote a report on this in the fall of 2021[2]. As for the case of decision trees our summary is taken from our previous work [3].

1. *Feed forward neural networks*

[2] We now introduce the basic concepts of the neural network. Specifically we are going to address a dense feed forward neural network, meaning the information is only moving forward trough the network and each node in a given layer is connected to each and every node in the following layer. The general structure of the network is sketched in figure 1. In the beginning we have an input layer containing one node for each feature of the data. The following layers are the so-called hidden layers which can have a custom chosen number of nodes for each layer. Finally we have the output layers with one node per target category.

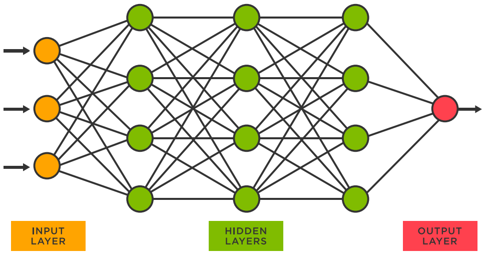


Figure 5: An illustrative image of dens feed forward neural network. Source, accessed 22.03.22

[2] Link to rapport describing neural networks is found here [2]
[3] Link to rapport describing decision trees and boosting is found here [3]

For a simple True and False type of problem a single output node which takes values between 0 and 1 is sufficient for the prediction. For classification between multiple classes we would require an output node for each class. The connection between the nodes is controlled by the introduction of the weights w and biases b. The weights and biases serve as the unknown parameter.

2. *Decision trees*

[3] In the regression case a decision tree divides the predictor space (the set of possible values $x_1, x_2, \dots, x_p$) into J distinct and non-overlapping regions $R_1, R_2, \dots, R_J$. For every new observation that falls into the region R_j we simply predict the mean value $\bar{y}_{Rj}$ of the training values in R_j. Since it becomes computational challenging to consider all possible partitions we are going to use a top-down approach. That is, we will consider the best split at each step rather than looking ahead and picking the splits that will lead to a better tree in the future. For a given feature j we consider a cut-point s which splits the predictor space into two regions

$$R_1 : \{X | x_j < s\},$$
$$R_2 : \{X | x_j \geq s\},$$

where X denotes all the features in the form of the design matrix. We want to choose the feature j and cut point s such that we minimize the MSE given as

$$\mathrm{MSE}(j, s) = \sum_{i: x_i \in R_1} (y_i - \bar{y}_{R1})^2 + \sum_{i: x_i \in R_2} (y_i - \bar{y}_{R2})^2$$

For N data points there is a maximum of $N - 1$ cut points which will make an unique contribution to the error. Thus for p features the total combination of splits is proportional to $p \times N$. Thus we can actually effort to go through all of them and pick the combination of j and s which minimize the MSE. After the first cut the tree branches into two regions for which we can repeat the splitting process again. We continue to split the branches until the max depth is reached, or when a certain regions contains less point than some predefinied limit. In all cases we have to stop splitting a certain branch when there is only one training point left in the region. We denote the end of a branch as a leaf.

3. *Gradient-boosted trees*

Gradient-boosting is a machine learning algorithm which uses a collective of "weak" classifiers in order to create one strong classifier. In the case of gradient-boosted trees the weak classifiers are a collective of shallow trees, which combine to form a classifiers that allows for deeper learning. As is the case for most gradient-boosting techniques, the collecting of weak classifiers is an iterative process.
We define an imperfect model $\mathcal{F}_m$, which is a collective of m number of weak classifiers, estimators. A prediction for the model on a given datapoint, x_i is defined as $\mathcal{F}_m(x_i)$, and the observed value for the aforementioned data is defined as y_i. The goal of the iterative process is to minimize some cost-function $\mathcal{C}$ by introducing a new estimator h_m to compensate for any error, $\mathcal{C}(\mathcal{F}_m(x_i), y_i)$. In other words we define the new estimator as:

$$\tilde{\mathcal{C}}(\mathcal{F}_m(x_i), y_i) = h_m(x_i), \tag{1}$$

where we define $\tilde{\mathcal{C}}$ as some relation defined between the observed and predicted values such that when added to the initial prediction we minimize $\mathcal{C}$.
Using our new estimator h_m, we can now define a new model as

$$\mathcal{F}_{m+1}(x_i) = \mathcal{F}_m + h_m(x_i). \tag{2}$$

The XGBoost [8] framework used in this analysis enables a gradient-boosted algorithm, and was initially created for the Higgs ML challenge. Since the challenge, XGBoost has become a favorite for many in the ML community and has later won many other ML challenges. XGBoost often outperforms ordinary decision trees, but what is gains in results it looses in interpretability. A single tree can easily be analysed and dissected, but when the number of trees increases this becomes harder.

D. Unsupervised Machine learning

1. *Anomaly detection*

Another tool used to do analysis is so called anomaly detection. In the event that one does not know what to look for, supervised learning such as boosting or ffnn's are quite slow and impractical. Anomaly detection is a way to find data that does not conform to what the system define as normal behavior.

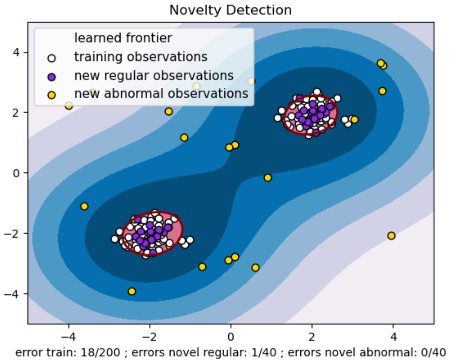


Figure 6: An illustrative image of anomaly detection using a one class support vector machine. Source, accessed 23.03.22

In figure 6 we see an example of a trained one class support vector machine that can separate new regular data from the abnormalities i.e anomalies. Other common examples could be fraud detection, intrusion detection, anomalous sensor data etc, which all have industrial purposes. An example, which is the one we will explore in this report, is anomaly detection in hadron collisions.
Suppose we were looking for new physics, such as new vector bosons like Z′ or W$^{\pm\prime}$[4], there are so many candidates for possible events that with supervised learning you have to guess at a candidate and hope to find that one. With anomaly detection on the other hand, we don't really assume what they look like, just that if it does not look like the background, for example the standard model, then is it some new physics. After that, one can try to find out with other tools what those events actually are.

In this report we are looking for the Higgs boson, which in 2011 and 2012 was to some extent to look for new physics. As explained in the previous sections, we have binary classification, thus we have the background events which will be our regular data, and the Higgs events which will be our anomalies.

2. *One class support vector machines*

Support vector machines is a method that tries to separate data using support vectors, and is done by maximizing the distance between the two categories by training supervised. This can either be done with linear data, or non linear. In the non linear case, one have to use the so called kernel trick, which maps the data to a high dimensional feature space to try and separate the data there. It can also be used with unlabeled data, where it tries to find natural clusters of data, and then map new data to these clusters.
We will in this report use the so called one class support vector machine, which will do a binary classification of our data. To understand this method we first take a look at a two class support vector machine. Suppose you have a dataset $\Omega = \{(x_0, y_0), (x_1, y_1), ..., (x_{n-1}, y_{n-1})\}; x_i \in \mathbb{R}^d$, where x_i is the i'th data point and $y_i \in \{-1, 1\}$ is the i'th output pattern. If the data is nonlinear, we use a kernel function to map the data to a higher dimentional feature space F where a "straight" line can we drawn to separate the data. When the data is mapped back to the original space, it will have the shape of a nonlinear curve, separated by a hyperplane.

Because the distance between the two closest points from the two classes to the hyperplane, the algorithm

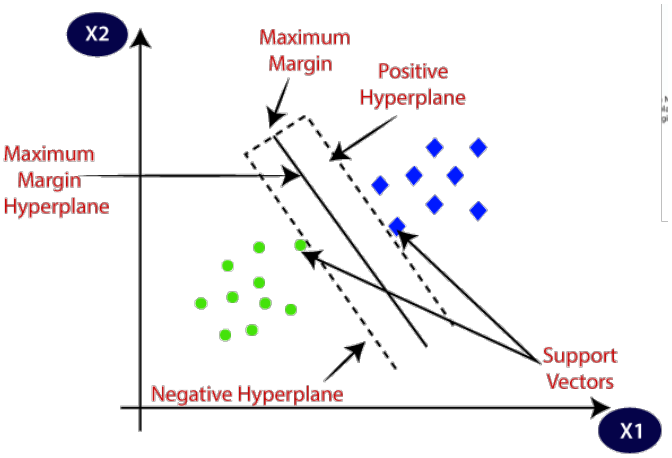


Figure 7: Diagram for linear classification using a support vector machine. Source, accessed 24.03.22

searches for the maximal margin between the two classes. To avoid overfitting we use a soft margin, thus some points will be allowed to live inside the margin. This is regulated by a constant $C > 0$ which regulates the trade off between the maximization of the margin and the amount of points that are allowed inside that margin. we thus formulate the SVM classifier with the given minimization formulation

$$min_{w,b,\xi_i}\frac{1}{2} \mid\mid w \mid\mid^2 + C\sum_{i=0}^{n-1}\xi_i,$$

which is subject to

$$y_i(w^T\phi(x_i) + b) \geq 1 - \xi,$$

for all $i = 0, ..., n-1$ and $\xi \leq 0$ for all $i = 0, ..., n-1$. Here ϕ is the transformed data points, ξ is the slack variable introduced to allow for soft margin. Using quadratic programming we end up with a final classification function [4], given as

$$f(x) = sgn\left(\sum_{i=0}^{n-1}\alpha_i y_i K(x, x_i) + b\right),$$

where K is the kernel function given as $K(x, x_i) = \phi^T(x)\phi(x_i)$, α_i are the Lagrange multipliers and the $sgn(\cdot)$ function is the sign function, returning in our case the class of the datapoint.

This leads us to one class support vector machines. Here the idea is that we instead of training on both positive (1 class) and negative (-1 class) we only train on one class of data. Instead we treat a part of the positive data as negative, and then try to define a boundary between the majority of the positive data points and the atypical data. What we then get is a somewhat similar minimization problem as above

$$min_{w,\xi_i,\rho}\frac{1}{2} \mid\mid w \mid\mid^2 + \frac{1}{\nu n}\sum_{i=0}^{n-1}\xi_i - \rho,$$

[4] More on candidates for possible CP restoration can be found here https://journals.aps.org/prd/pdf/10.1103/PhysRevD.12.1502 and https://link.springer.com/content/pdf/10.1007/BF01556677.pdf

that is subject to $w \cdot \phi(x_i) \geq \rho - \xi$ for all $i = 0, ..., n-1$ and $\xi_i \geq 0$ for all $i = 0, ..., n-1$[6]. Here ν is a parameter that makes two contributions to the solution:

- Set an upper boundary on fraction of anomalies (outliers)
- Set a lower boundary of training samples used as support vectors

From this we get a classifier function given as

$$f(x) = sgn\left(\sum_{i=0}^{n-1} \alpha_i K(x, x_i) - \rho\right)[6].$$

3. *Autoencoders*

Auto Encoders are a subgroup of feed forward neural networks. The goal of an auto encoder is to compress the information sent in into a few variables, also known as the latent space, and then decode that information back. This allows the algorithm to learn the most important features of the data, and is shown in the figure below.

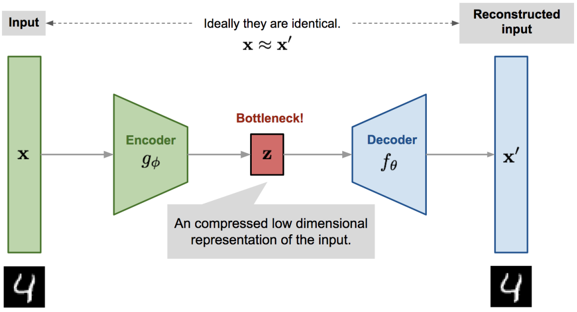

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
x
Encoder
g_ϕ
Bottleneck!
z
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.

Figure 8: Diagram for auto encoder architecture, using image reconstruction as example. Source, accessed 23.03.22

Some uses of this are image recognition, denoising or anomaly detection. What is common for these three methods is the concept of reconstruction. The auto encoder first deconstructs the data that is put in, reducing the number of nodes for each layer, until it reaches the latent space, depicted in figure 8 as **z**. From here the decoder reconstructs the information, increasing the nodes per layer. Note that the number of nodes and layers for the decoder and encoder does not need to be the same. We only require that the input and output layer of the complete model is of the same shape. We thus end up with a reconstruction **x'**. Now, to train the model, we parameterize the individual parts of the algorithm as such. The deconstructed information is given as

$$\mathbf{z} = g_\phi(\mathbf{x}),$$

and reconstructed information is given as

$$\mathbf{x'} = f_\theta(g_\phi(\mathbf{x})),$$

where the parameters (θ, ϕ) are learned to to output the reconstructed data as close to the original information as possible, i.e

$$\mathbf{x} \approx f_\theta(g_\phi(\mathbf{x})).$$

To reach as small as possible error, we use the mean squared error

$$L_{AE}(\theta, \phi) = \frac{1}{N}\sum_{i=0}^{N-1}\left(\mathbf{x}^i - f_\theta(g_\phi(\mathbf{x}^i))\right)$$

4. *Isolation forest*

Isolation forests was first proposed in 2007[5] as a method for anomaly detection. It is an ensemble method, and is based on the same principles as those mentioned in section II C 2. The basis for the algorithm is that outliers tend to more easily separate from the data, compared to inliers. This is shown in the figure below.

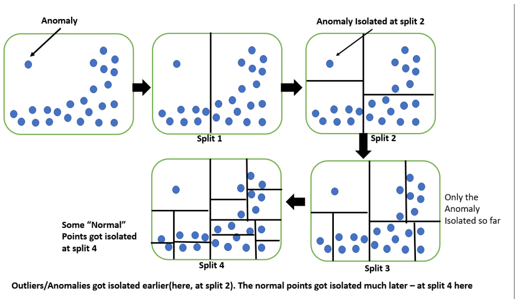

Anomaly
Anomaly Isolated at split 2
Split 1
Split 2
Only the Anomaly Isolated so far
Some "Normal" Points got isolated at split 4
Split 4
Split 3
Outliers/Anomalies got isolated earlier(here, at split 2). The normal points got isolated much later – at split 4 here

Figure 9: Cuts done to separate the outlier from the inliers. Source, accessed 26.04.22

In this figure we see an ideal case, where the anomaly is separated from the rest of the data. The isolation forest model does not however base its classification on a measure of distance or density, but rather, how many splits do you need to completely isolate the data. In figure 9 the anomaly took 2 splits to isolate the anomaly, whilst the more grouped data took at least 4 splits.

To create these splits, the algorithm selects a random feature of the data, in the case of the figure above, either the x or y axis. It then picks a value between the minimum and maximum value allowed for the given feature and makes a split. It is important that the pick of feature and split value is randomly picked from a distribution. This allows for an average value of number of splits for an anomaly to be created using multiple trees.

To assert whether or not an event is an anomaly, a score is created associated with that event. Liu, Fei Tony et al.[5] proposed the following scoring function

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}. \quad (3)$$

Here $E(h(x))$ is the expected number of splits for the given event and $c(n)$ is a normalization constant. The normalization constant is given as

$$c(n) = 2H(n-1) - (2(n-1)/n),$$

where $H(i)$ is the harmonic number, and can be estimated by $ln(i) + 0.5772156649$, and n is the number of instances in the dataset[5]. Because s is monotonic to $h(x)$, i.e $E(h(x)) \rightarrow 0, s \rightarrow 1$ and $E(h(x)) \rightarrow n-1, s \rightarrow 0$, we can define the scores such that

- if an event returns s close to 1 it is an anomaly
- if an event returns s less than 0.5 it is safely regarded as an inlier
- if all instances return s around 0.5, the entire sample does not really contain any distinct anomalies

E. Method evaluation

1. Accuracy

In particle physics one rarely talks about individual events. This is due to the fact that no significant result can be derived from just one event, but instead requires a large set of result as well as reproducibility. Therefore, most of the results in this rapport will not be based on accuracy, as this will simply be comparing many individual result.

2. ROC-curve

A ROC curve, or receiver operating characteristic curve is a tool used to measure and visualize a binary classifiers ability to predict trends. The curve is plotted on x,y-axis where the x-axis represents true positive rates and the y-axis represents false positive. The different values for the curve are calculated with different thresholds, i.e the value deciding whether an event is 1 or 0, signal or background.
If a classifier has learned nothing and is simply guessing, the ROC curve will be a linear curve going from 0 to 1. This line is often drawn in ROC curve. The better the classifier is, the higher the ROC curve will bend towards the upper-left corner of the graph. The worse it is, the more it will bend to the lower right corner. An example of all that has been discussed can be seen in figure (10).

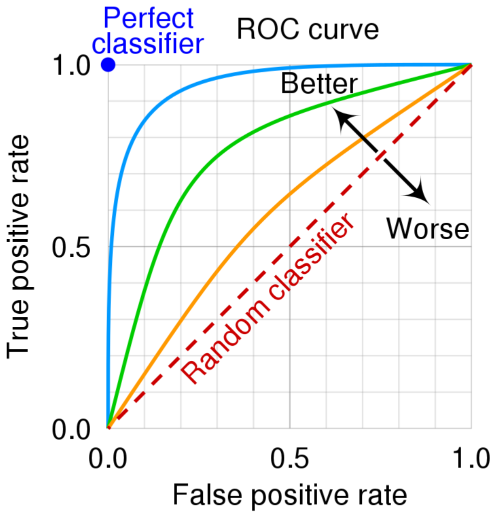


Figure 10: An illustrative image of a ROC curve. Source, accessed 19.03.22

3. Distribution histograms

As was previously mentioned in II E 1, one rarely talks about definite predictions of individual events in particle physics. Instead we try to create classifiers that are able to separate entire distributions. As we are in the fortunate position of having labeled training data, we can not only plot the distribution of the output for the different classifiers, but also color mark the actual label of the different distribution. This way we can visualize the distribution of the output as well as the accuracy of the distribution. As a final perk, it will additionally give us some insight into the the ideal values for the threshold, ϵ.

4. AMS

Given that the test-data is not labeled and, as we will see later there is much overlap we must use alternative methods when evaluating a classifier's output. The way one usually deals with such problems is that one begins by assuming that the data only contains background. Then, using a classifier we investigate whether the dataset contains signal events in addition to the background. Finally we apply a statistical test to determine the validity of rejecting the background-only hypothesis. In compliance with the competition description, the statistical test for this analysis will be the AMS, or approximate median significance.
In simple terms the AMS measures the significance of the classifier discovering a certain amount of signal events. The derivation of the AMS formula is based on a event distribution for a given region in the feature space. To arrive at the AMS we start by assuming a Poisson dis-

tribution given as

$$P\left(n \mid \mu_{\mathrm{s}}, \mu_{\mathrm{b}}\right)=\frac{\left(\mu_{\mathrm{s}}+\mu_{\mathrm{b}}\right)^{n}}{n!} e^{-\left(\mu_{\mathrm{s}}+\mu_{\mathrm{b}}\right)}, \qquad (4)$$

where n is the total number of events and μ_s and μ_b are the expected number events for the signal and background. Next we are interested in calculating the likelihood ratio between the background-only hypothesis and the background and signal, i.e μ_s = 0 and μ_s > 0.

$$\lambda=\frac{P\left(n \mid 0, \mu_{\mathrm{b}}\right)}{P\left(n \mid \mu_{\mathrm{s}}, \mu_{\mathrm{b}}\right)}=\left(\frac{\mu_{\mathrm{b}}}{\mu_{\mathrm{s}}+\mu_{\mathrm{b}}}\right)^{n} e^{\mu_{\mathrm{s}}}=\left(\frac{\mu_{\mathrm{b}}}{n}\right)^{n} e^{n-\mu_{\mathrm{b}}}, \qquad (5)$$

where we have used that $\mu_{\mathrm{s}}=n-\mu_{\mathrm{b}}$.
Further we introduce Willk's theorem [5], which lets us define statistical significance, $\mathcal{Z}$ using λ as

$$\mathcal{Z}=\begin{cases}\sqrt{-2 \ln \lambda} & \text{if } n>\mu_{\mathrm{b}} \\ 0 & \text{otherwise}\end{cases}. \qquad (6)$$

By inserting our λ from eq.(5) we find

$$\mathcal{Z}=\sqrt{2\left(n \ln \left(\frac{n}{\mu_{\mathrm{b}}}\right)-n+\mu_{\mathrm{b}}\right)} \qquad (7)$$

As we do not know μ_b, μ_s we instead introduce two variables s and b. s and b are defined as the luminosity normalized true and false positive rates. By defining n=s+b and μ_b = b we find

$$\mathrm{AMS}=\sqrt{2\left(\left(s+b+b_{\mathrm{reg}}\right) \ln \left(1+\frac{s}{b+b_{\mathrm{reg}}}\right)-s\right)}. \qquad (8)$$

b_{reg} was introduced for the Challenge in order to reduce the variance of the AMS, and is set to 10.
Given that the data is produced with Monte Carlo simulation, the number of s- and b-events can be whatever the simulator wants it to be. For the purpose of training a classifier there have been added a much larger amount of signal events than what is realistically expected. In reality one expects only a hand-full of signal events per million of background. To deal with this fact each event is given a weight, w_i where the weight acts as a probability. Using the weights we can define s and b as the following

$$s=\sum_{i \in \mathcal{S} \cap \widehat{\mathcal{G}}} w_{i} \quad \textit{and} \quad b=\sum_{i \in \mathcal{B} \cap \widehat{\mathcal{G}}} w_{i}, \qquad (9)$$

where $\widehat{\mathcal{G}}$ is the region of indexes of events the classifier predicted as signal event, and $\mathcal{B}$ and $\mathcal{S}$ are the regions where the events in fact are background and signal. In this way summing the weights gives a more realistic approximation of the distribution of signal and background events.

III. IMPLEMENTATION

The numerical implementations can be found on our github.

A. Data handling

As stated earlier in the report, the data we are using is provided by Atlas and is produced with Monte Carlo simulations. All together the training data consists of 250 000 events, where each event has 30 features [6]. The values of each features varies and can be both binary of continues. The signal events were generated with a fixed mass of the Higgs of 125GeV, and the background was generated with the three processes discussed in section II B.
Given the attempt to simulate data as realistic as possible, some measurements have been effected by imperfect measurement tools. In other words, the Monte Carlo simulations have added events with unrealistic values. This "bad" measurements have all been given the value of *NaN* for the sake of this analysis. Another point to make is that in order to preserve the laws of physics, some events **have** to put *NaN* is some categories. An example would be that a feature would hold the transverse momentum of the third jet. If the physical event does not have 3 jets, then you cannot put a value there, or you break the laws of physics. The effect of this *NaN* values will be different for each method. How we treat these "bad" measurements or indeed the data in general is therefore crucial for the analysis.
In our analysis we have tested several methods of handling our data. These methods are listed as the following:

- $standardScale()$
- $setNanToMean()$
- $fillWithImputer()$
- $removeOutliers()$
- $removeBadFeatures()$

[5] S. S. Wilks. The large-sample distribution of the likelihood ratio for testing composite hypotheses. Annals of Mathematical Statistics, 9:6062, 1938.

[6] A table of all features is presented in the appendix.

Each function somehow effects the data, and would be applied before any ML method. The *standardScale* scales each feature independently by subtracting the average of the mean and dividing by the standard deviation. *setNanToMean* is used to handle the case of "bad" measurements, by replacing each NaN-value by the mean of the feature. $fillWithImputer$ tackles the same issue, but uses a ML algorithm to fill in all "bad" measurements. *removeOutliers* removes all events with any features outside a given number of standard deviations. This function is used in the case of the ML method showing great sensitivity to outliers. Lastly we the function *removeBadFeatures* is used to remove a feature from all events if that feature has more than a given percentage of "bad" measurements.

B. The algorithms

In the following section we will present the different data handling and specific implementations used for each ML method. In general all method have been tested with different data handling functions **??**, then used the data to preform a grid search to obtain optimal parameters and finally we apply the optimized method on the test data.

1. *Neural network*

In the neural network analysis we are forced to somehow remove/replace any NaN-values, given the algorithm behind the method. Any NaN value sent into the neural network would quickly propagate through to each node and all results would be comprised. This being the case we therefore tested for both the *setNanToMean* function and the $fillWithImputer$. In addition we scaled the data using the function *standardScale*.
To optimize our parameters we used the $keras - tuner$. The tuner was used to optimize number of layers as well as number of nodes per layer, learning rate and activation functions.

2. *XGBoost and Decision trees*

Given that in both decision trees and in the XGBoost method each features is treated individually, there was no need for scaling. But, where the XGBoost differs from the normal decision trees is that it is not effected by the NaN- values. Therefore the Decision tree algorithm uses the $fillWithImputer$, where the XGBoost does not need to.
As for optimization both methods use scikitLearn's GridSearchCV. In the case of decision trees the only variables that showed any effect of optimizing were max_depth and $min\_samples\_leaf$. In the case of XGBoost three variables were tested during the optimization; $n\_estimator$, max_dept and eta.

3. *Autoencoders*

As with the neural networks, we needed to handle the datapoints, as some features had NaN values. We chose here to simply set the NaN values to the mean value of the feature, and then scale the dataset with standard scaling.

To optimize the network we also here used Keras tuning. For node optimization we set varying intervals that got had somewhat similar number of different values, but ensured that the layer above always had at least one more node. This way we get the "dumbbell" shape on the network. For activation function optimization we gave the tuner the choice between ReLU, LeakyReLU and Tanh, whilst the output layer also had the option of sigmoid. The learning rate were tested for the values $[9 \cdot 10^{-2}, 9.5 \cdot 10^{-2}, 1 \cdot 10^{-3}, 1.5 \cdot 10^{-3}]$, and the optimizer used was ADAM.

IV. RESULTS AND DISCUSSION

A. Trends in the data

The hope of the analysis is that there exists a sufficient trend in the features of the Higgs data that will allow us to separate signal from background. The spread of values for A sample of features are plotted in figure 11. It is obvious from the figure that there are some features that exhibit clear trends and some that don't. As expected the invariant mass of the particle has a clear difference in peak, but there is also some overlap. This means that any effective algorithm should hope to rather easily attain some knowledge on the data, but given the amount of overlap and weak trends for some of the features it could prove hard to surpass a given accuracy.

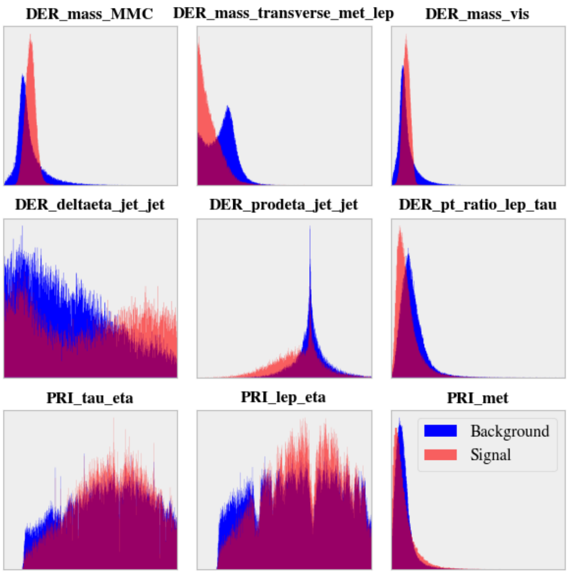


Figure 11: A visualisation of the spread of values for 9 given features. Blue representing background and red representing signal.

To further inspect the difference in value distribution of the different features we made a boxplot presented in figure 12. The figure displays several features where the signal and the background equal in both median and distribution. But, as was evident from figure 11, we observe equally many features where there are subtle differences. For features like the invariant mass and the transverse mass there are clear differences in both median and distribution with little overlap. For features like the total transverse energy and the ratio of the transverse momenta of the lepton and the hadronic tau, there are are difference in median but also a lot of overlap. Taking advantage of both, and other differences in trend is crucial when we try to train different networks.

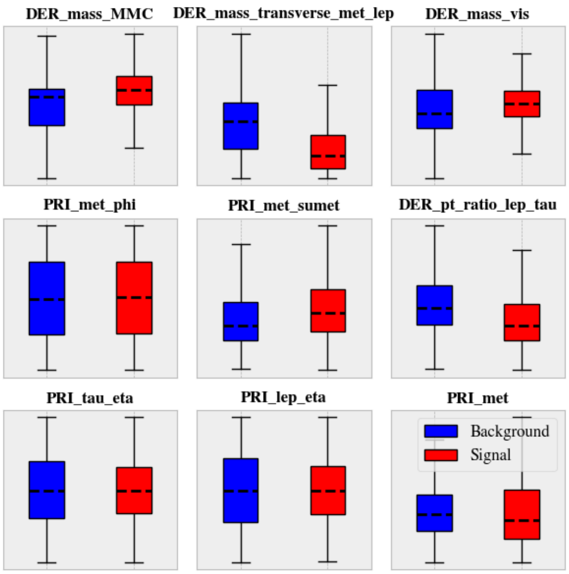


Figure 12: A boxplot for 9 given features, comparing background (blue) signal (red).

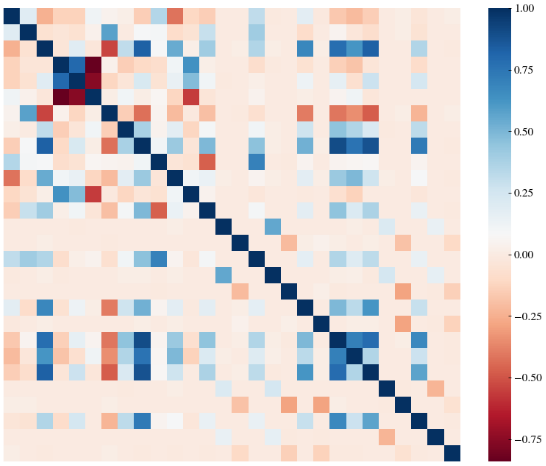


Figure 13: A correlation plot for all 30 features. The features are ordered 0-30 from up to down and left to right, where the order of the features are presented in the appendix.

B. Grid search

1. *Neural network*

In the grid search we found the ideal number of hidden layers to be 6; 5 dense and 1 dropout. The dropout layer was added after the optimizer, as an attempt to minimize over-fitting due to the large number of parameters. The ideal structure of the layers were found to be the following:

where each dense layers uses a leakyRelu activation

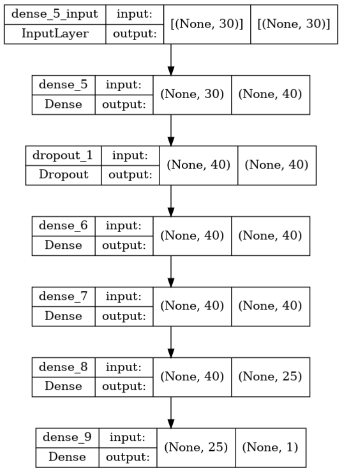


Figure 14: Neural network layout after Keras Tuner.

function with a threshold of 0.1. The ideal learning rate was found to be 0.0015.

2. *Decision trees and XGB*

To preform the grid search for the decision tree and XGBoost algorithms we applied scikit-learn's GridSearchCV. Through many tests and trials we found that the ideal parameters for the decision tree were the default values [7] but for two parameters. Through the grid search we found that the remaining optimal parameters were; max_depth : 9 and $min\_samples\_leaf$: 7.
In the case of XGBoost all but three optimal parameters were found to be the default. The remaining ideal parameters were found to be; max_depth : 1, $nr\_estimators$: 50 and eta : 0.1.

3. *Auto-encoders*

In the Keras Tuner gridsearch for the Auto encoder we found that three hidden layers for the encoder and decoder was optimal. Here we found the optimal learning rate to be 0.0015, and the table below show the hidden layers and the ideal activation function.

Table I: Table showing the layers of the model components, and the activation function for the layer.

Model part	Layer number	Activation function
Encoder	Hidden 1	Tanh
	Hidden 2	LeakyReLU
	Hidden 3	LeakyReLU
Decoder	Hidden 1	LeakyReLU
	Hidden 2	Tanh
	Hidden 3	LeakyReLU

4. *Isolation forest*

The cluster algorithm is used with the standard hyperparameters set by Scikit-Learn. Even though the algorithm is based on decision trees, quite heavily, it does require non-NaN values, so that it can make the necessary splits for the features. Thus, for datahandling, the data removed all the NaN values and inserted the mean of the given feature. We did not perform any scaling of the data before clustering.

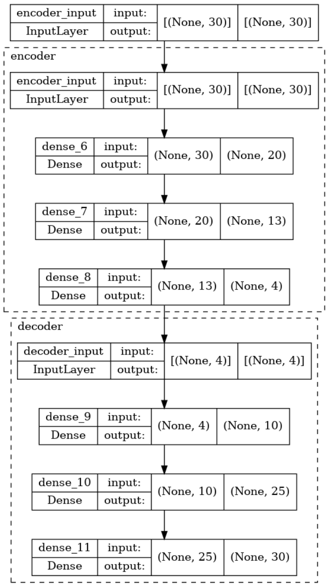


Figure 15: Aotu Encoder layout after Keras Tuner.

In the figure above we see the complete model structure after tuning with Keras. The tuning was done on a Macbook Pro M1 laptop, and took 2 minutes and 17 seconds, yielding a best validation MSE of 0.0254. The run was done with a batchsize of 4000, and with a max training epoch range of 50 epochs.

C. Performance on training data

In this section we will compare the output distribution and ROC-curve of all the ML techniques. All the models were trained on a portion of the training data (80%) and all results were created with the remaining data(20%), a validation set.

1. *Neural network*

In figure 16 we plotted the history of the Neural network over 500 epochs, both the accuracy and the loss. The network quickly converges and after a 100 epochs both the loss and the accuracy oscillate around the minimum values.

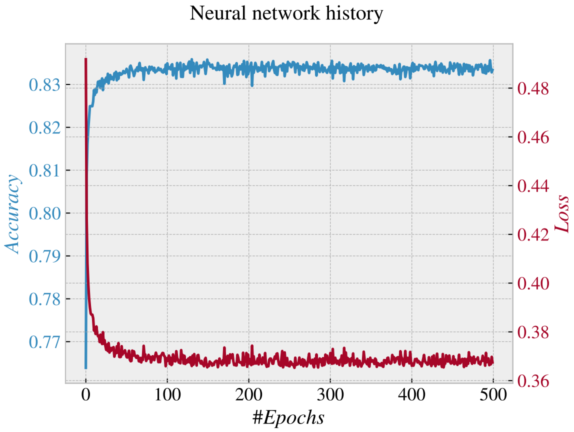


Figure 16: A plot of the training history for the neural network over 100 epochs. All parameters were for the network were decided in the grid search from IV B 1

In figure 17 we display the output distribution of the neural network. In the figure we observe a clear separation of the background and signal. Most of the background data has been separated to the interval of [0,0.5] and the the signal has likewise been separated to [0.5, 1]. The extent of the separation is made clear through the differences in mean value, 0.49.
Another interesting observation it the median value of both the signal and background. The difference in median is 0.99. This means that the neural network has placed most background values to be 0 and most signal to be 1. In other words the network shows great confidence in its predictions. This could however be a sign of over-fitting.
In figure 18 we plot the ROC-curve of our network. From the figure we observe that the model has greatly curved towards an optimal model ([0,1]). The curve creates an areal of 0.91, which shows sign of a good classifier, but with room for improvement.

2. *Decision trees*

In figure 19 we plotted the output distribution of the decision tree. Immediately we observe a far less systematic spread than for the neural network. This is because of the underlying algorithm of each method. The complexity of the neural network allows for a much larger range of values. In the case of using only one decision tree with a limited depth, only certain realistic values are accessible.
As for the separation of mean and median, the results for the decision tree is worse tha for the neural network. The difference in median was 0.82 and in mean was 0.45. This indicated that the decision tree preformed worse than the neural network.
In figure 20 we plotted the ROC-curve of the decision tree. Again we observe that the decision tree preformed worse than the neural network. The decision tree only achieved an area of 0.88.

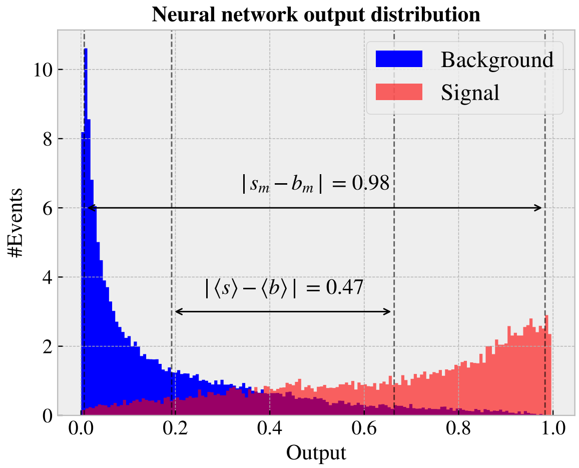


Figure 17: A plot of the output distribution for the neural network after 500 epochs. The distance between the mean and median for s and b is visualized in the center of the plot.

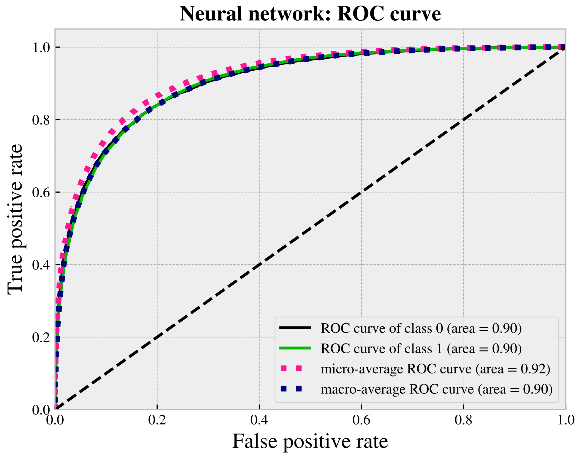


Figure 18: A plot of the ROC-curve for the neural network after 500 epochs.

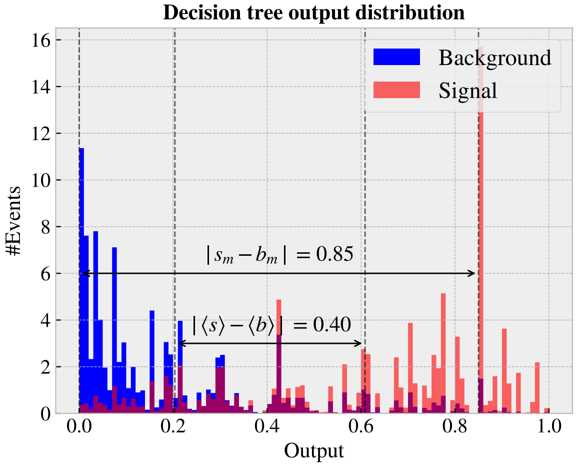


Figure 19: A plot of the output distribution for the decision tree algorithm. The distance between the mean and median for s and b is visualized in the center of the plot.

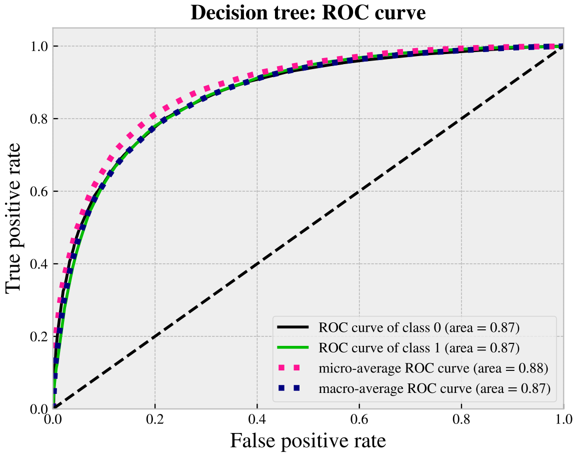


Figure 20: A plot of the ROC-curve for the decision tree algorithm.

3. *XGBoost*

In figure 21 we plotted the output distribution of XGBoost. Figure 21 much resembles the neural network, but for the median of the signal. We observe that the peak of the signal no-longer is located around 1.0. This results in the decrease in distance of median. On the other hand the XGBoost has increased the distance in mean, thereby outperforming the neural network ever so slightly, 0.01.
The XGBoost also differs from the spread of the decision tree. This is a consequence of the large collective of weak classifiers, which allow for a far larger range of realistic values.

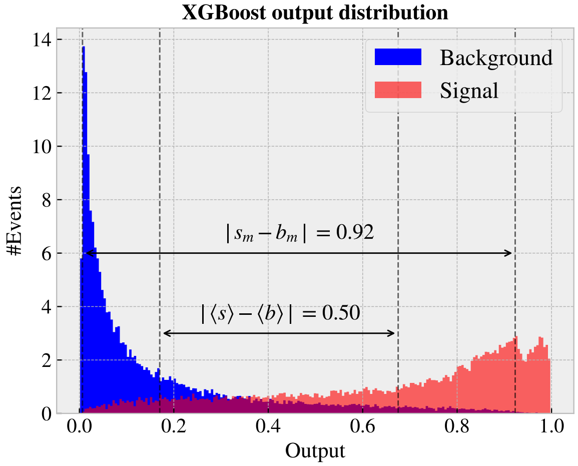


Figure 21: A plot of the output distribution for the XGBoost model after 500 epochs. The distance between the mean and median for s and b is visualized in the center of the plot.

In addition it is interesting to note that the mean for the background is closer to 0 than for the neural network, but the mean for signal is further away from 1. It seems from the two figure, 21 and 17 that the two mean values have shifted to the left. This is evidence that the XGB seems to better classify background than signal. Since the training distribution of background and signal is unrealistically close to even, it could explain why the accuracy of the XGBoost is relatively close to the neural network. In the AMS (to be discussed in section IV D) score on the other hand, the unrealistic distribution of signal and background is dealt with through the summing of weights, and would therefore explain why the XGBoost is that much more dominant.

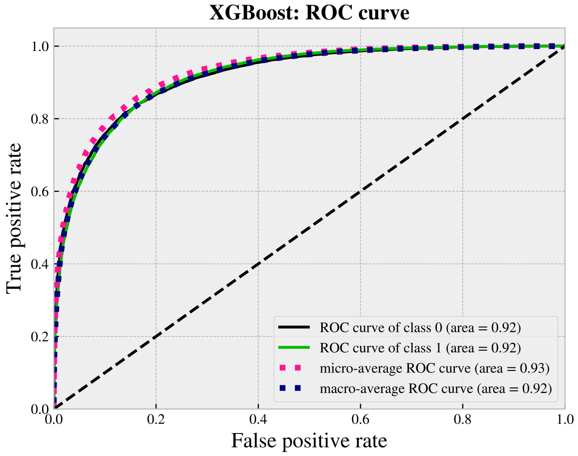


Figure 22: A plot of the ROC-curve for the XGBoost.

Finally we study the ROC-curve of the XGBoost, figure 22. The figure displays that the XGBoost better classifies the data than both the neural network and the decision tree, achieving an area of 0.92. Thereby beating the neural network and the decision tree by 0.01 and 0.04 respectively.

4. *Autoencoder*

In figure 23 we see that the autoencoder struggles to separate the distributions of signal and background based on the MC data. We also see in figure 24 that all though the model struggles to separate the two distributions, there is a "better than guessing" trend for the model.

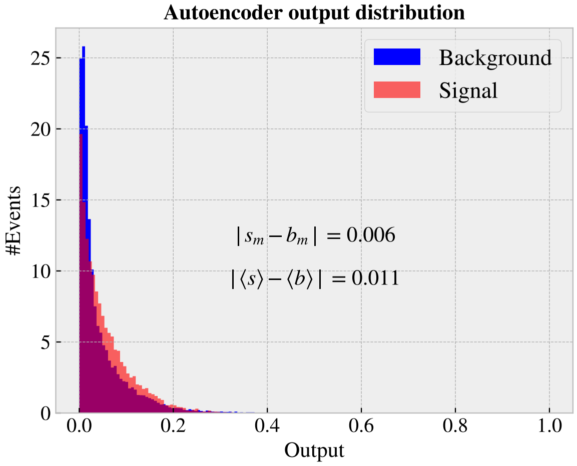


Figure 23: A plot of the output distribution for the Autoencoder model after 50 epochs.

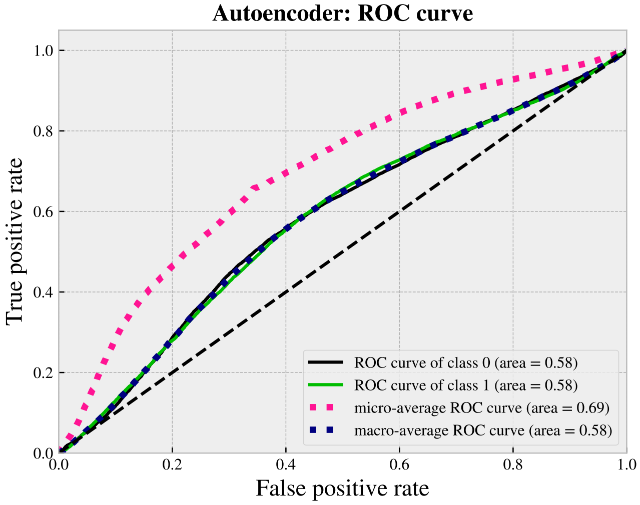


Figure 24: A plot of the ROC-curve for the Autoencoder model.

Below we show the error distribution of the model classification, but the data here is tweaked. We removed here the events where if one or more features was more than $\pm 5\sigma$ from the mean over all events for that given feature(s).

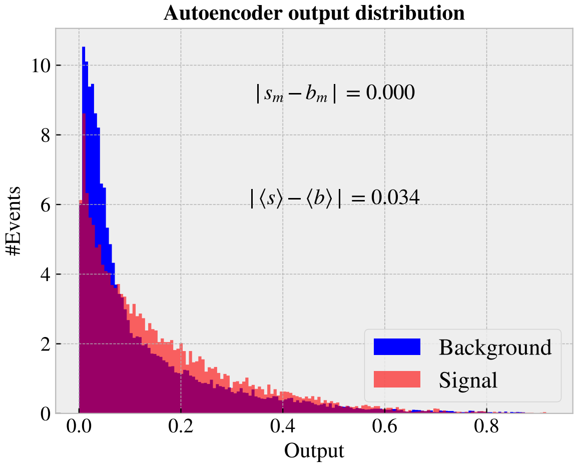


Figure 25: A plot of the output distribution for the Autoencoder model after 50 epochs, having removed outliers of 5 sigma.

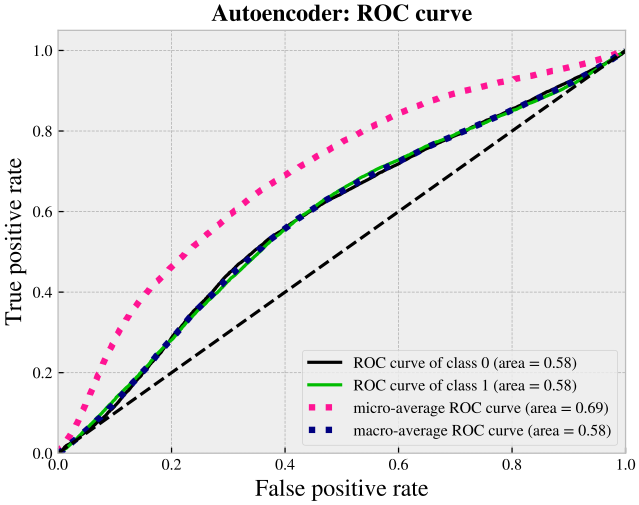


Figure 26: A plot of the ROC-curve for the Autoencoder model.

By removing these outliers, we see in figure 26 and figure 25 that the separation of the mean more than doubles. The difference in median is contributed to the new intervals for the bins in the histogram. The trend in the data is still the same although easier to spot. Although the autoencoder achieves a far worse result than our supervised methods, we do see a clear difference in trend for the signal and background. This means the autoencoder has in fact learned something.

5. *Isolation forest*

The results of the clustering are shown in figures 27 and 28. In figure 27 we see the anomaly score outputed by the algorithm. The score calculated by scikit learn is, as themselves put it, the opposite of what is written in Liu, Fei Tony et al.[5], and so to get a score that makes more sense, we normalized the output, and subtracted it from 1. We see here from the definitions of the anomaly score, that the classifier claims there are very few anomalies. This is expected, as the data is very similar, as mentioned in section IV A, and the number of events are relatively small. One could suggest other sets of hyper parameters, but we cannot do any tuning to claim it a truly unsupervised method. To tune the parameters would mean exposure to the labels of the training data, and then it cannot be called an unsupervised method anymore. There is also the possibility that some handling of the data, for example by tweaking the categories in such a way that we did not need to have *NaN* values, could help the classifier to further separate the signal from background.

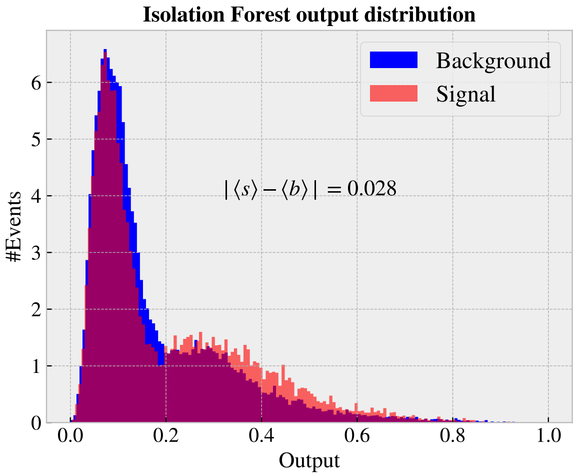


Figure 27: A plot of the output distribution for the Isolation forest model.

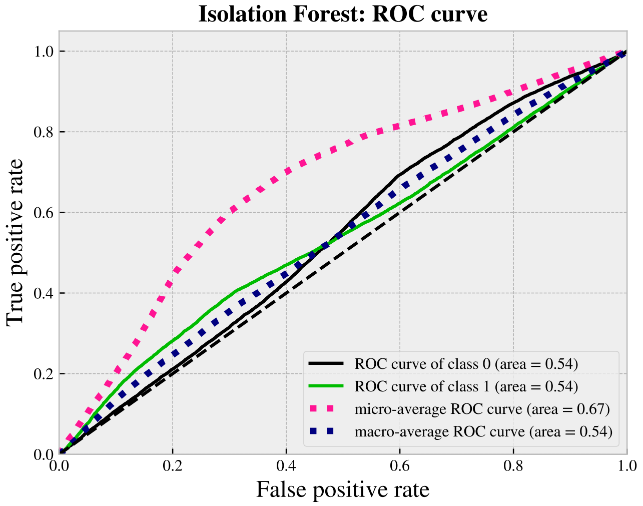


Figure 28: A plot of the ROC-curve for the Isolation forest model.

In figure 28 we an interesting trend. The ROC curves of class 0 and 1 change around 0.4 around the macro average ROC curve. This fits somewhat well with what we see in figure 27, where we see a sudden spike around 0.4. Still with these slight fluctuations, we see that the Isolation forest preforms very similarly with the autoencoder.

D. Comparison

In this section we will compare the results on both the training data and the test data. Firstly the comparison we be made between the supervised and semi/unsupervised methods seperatly, then finally a comparison will be made between all method. In this section you will also find the AMS results for all the methods **??**.

1. *Supervised methods*

- Results on training data.
- Results on test data.
- Interpretability: Decision trees are easiest to read, then boosted decision trees, then neural networks. What boosted decision trees gain in results, it looses in interpretability.
- Bias? For networks and support-vector machines we added mean or used imputer. This could result in un-physical events. In further research this would be interesting to fix.

2. *(Semi)-Unsupervised methods*

- difference with tuning
- results on training and test
- interpretability

In the case of (semi)-unsupervised models, there are multiple differences that are worth discussing. First of all, the two methods are fundamentally two different approaches, where the autoencoder bases its separation on reconstruction error, whilst the isolation forest bases its separation on a anomaly score derived from average splits. We saw that even though the separation distributions and the ROC curves where quite similar, the Isolation forest model got 3 times as high AMS score, as shown in table**??**. There could be multiple reasons for this, but we suspect that the reconstruction model needed more data to train on.

There is also the issue with tuning. The autoencoder was tuned using Keras tuner, but because the problem what framed as a reconstruction problem, you can tune the algorithm without a priori knowledge of the labels. This is not possible with the isolation forest model, and one can speculate whether the algorithm could get even higher AMS score, given more ideal hyperparameters. Then again, the model would turn into a semi unsupervised model.

There is also the issue of interpretability. With the isolation forest model, all of the calculations are done via an API with little to no oversight over what goes on. The autoencoder however, all though written in Tensorflow, is designed down to almost every last detail, with full oversight of the layers, nodes, activationfunctions etc.

3. *Collective comparison*

It is important to note that the datahandling for the neural network, autoencoder, decision tree and isolation forest all needed to medigate the features with *NaN* values. A typical solution amonst data scientists and alike is to set all *NaN* values to the mean of the feature, but this can be somewhat problematic. For stock marked analysis or house prize analysis, having a feature put to a mean value is not necessarily problematic, all though it could make little sense for us. A house without a garage where a feature told what color the garage has, would have to put *NaN* in the slot, as the house is without the garage, and thus the color of it. For the analysis of housing prices it could very well be a factor, but for the algorithm it might not make any difference if the house has it or not, given all the other features.

In physics however, where we try to discover known and unknown particles, we have to ensure that we obey the laws of physics, because if not, the analysis could claim to find a particle having broken the laws of physics. For this report we chose to not consider this as a problem, as the purpose of the report is to try different algorithms and compare them, and not to do actual physics research. It is however important to be aware of, and could have been done. One idea to avoid the mean setting could be to change some of the features, by reforming the information. Let's say you have three features that hold the transverse momentum of the first, second and third lepton. In the event that you only measure two leptons, we could rewrite the three features as one, number of leptons, and assign 2. This is a bit crude method, and there are other ways to avoid the mean setting.

Table II: Ouput separation and ROC score for all methods

Method	Mean separation	ROC score
Neural network	0.49	0.91
XGboost	0.50	0.92
Decision trees	0.46	0.89
Autoencoder	0.01(0.03)	0.58
Isolation forest	0.028	0.54

=2ex

Table III: AMS for the test data using different methods.

Method	AMS
Neural network	3.30
XGboost	3.56
Decision trees	2.83
Autoencoder	0.33
Isolation forest	1.01

V. CONCLUSION

REFERENCES

[1] Georges Aad et al. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Phys. Lett. B* 716 (2012), pp. 1–29. DOI: 10.1016/j.physletb.2012.08.020. arXiv: 1207.7214 [hep-ex].

[2] Sakarias Frette, William Hirst, and Mikkel Metzch Jensen. "A computational analysis of a dense feed forward neural network for regression and classification type problems in comparison to regression methods". In: (2022), p. 5. URL: https://github.com/Gadangadang/Fys-Stk4155/blob/main/Project%202/article/Project_2_current.pdf.

[3] Sakarias Frette, William Hirst, and Mikkel Metzch Jensen. "Bias variance analysis". In: (2022), p. 3. URL: https://github.com/Gadangadang/Fys-Stk4155/blob/main/Project%203/article/Project_3_current_part2.pdf.

[4] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. "Kernel methods in machine learning". In: *The Annals of Statistics* 36.3 (2008), pp. 1171–1220. DOI: 10.1214/009053607000000677. URL: https://doi.org/10.1214/009053607000000677.

[5] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation Forest". In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[6] Bernhard Schölkopf et al. "Support Vector Method for Novelty Detection". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper/1999/file/8725fb777f25776ffa9076e44fcfd776-Paper.pdf.

[7] Scikit-learn developers. *DecisionTreeRegressor*. 2022. URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html.

[8] The XGBoost Contributors. *XGBoost*. Version 1.5.2. URL: https://xgboost.readthedocs.io/en/stable/#.

Appendix A: Features

Table IV: Features and their description

Feature	Description
DER_mass_MMC	The estimated mass mH of the Higgs boson candidate, obtained through a probabilistic phase space integration (may be undefined if the topology of the event is too far from the expected topology)
DER_mass_transverse_met_lep	The transverse mass between the missing transverse energy and the lepton.
DER_mass_vis	The invariant mass of the hadronic tau and the lepton.
DER_pt_h	The modulus of the vector sum of the transverse momentum of the hadronic tau, the lepton, and the missing transverse energy vector.
DER_deltaeta_jet_jet	The absolute value of the pseudorapidity separation (22) between the two jets (undefined if PRI jet num ≤ 1).
DER_mass_jet_jet	The invariant mass (20) of the two jets (undefined if PRI jet num ≤ 1).
DER_prodeta_jet_jet	The product of the pseudorapidities of the two jets (undefined if PRI jet num ≤ 1).
DER_deltar_tau_lep	The R separation (23) between the hadronic tau and the lepton.
DER_pt_tot	The modulus of the vector sum of the missing transverse momenta and the transverse momenta of the hadronic tau, the lepton, the leading jet (if PRI jet num ≤ 1) and the subleading jet (if PRI jet num = 2) (but not of any additional jets).
DER_sum_pt	The sum of the moduli of the transverse momenta of the hadronic tau, the lepton, the leading jet (if PRI jet num ≤ 1 1) and the subleading jet (if PRI jet num = 2) and the other jets (if PRI jet num = 3).
DER_pt_ratio_lep_tau	The ratio of the transverse momenta of the lepton and the hadronic tau.
DER_met_phi_centrality	The centrality of the azimuthal angle of the missing transverse energy vector w.r.t. the hadronic tau and the lepton $C = \frac{A+B}{\sqrt{A^2+B^2}}$, where $A = \sin(\phi_{met} - \phi_{lep})$, $B = \sin(\phi_{had} - \phi_{met})$, and ϕ_{met}, ϕ_{had} and ϕ_{lep} are the azimuthal angles of the missing transverse energy vector, the lepton, and the hadronic tau, respectively.
DER_lep_eta_centrality	The centrality of the pseudorapidity of the lepton w.r.t. the two jets (undefined if PRI jet num ≤ 1) $\exp\left[\frac{-4}{(\eta_1-\eta_2)^2}\left(\eta_{lep} - \frac{\eta_1+\eta_2}{2}\right)^2\right]$, where η_{lep} is the pseudorapidity of the lepton and η_1 and η_2 are the pseudorapidities of the two jets. The centrality is 1 when the lepton is on the bisector of the two jets, decreases to 1/e when it is collinear to one of the jets, and decreases further to zero at infinity.
PRI_tau_pt	The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the hadronic tau.
PRI_tau_eta	The pseudorapidity η of the hadronic tau.
PRI_tau_phi	The azimuth angle ϕ of the hadronic tau.
PRI_lep_pt	The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the lepton (electron or muon).
PRI_lep_eta	The pseudorapidity η of the lepton.
PRI_lep_phi	The azimuth angle ϕ of the lepton.
PRI_met	The missing transverse energy $\mathbf{E}_T^{miss}$.
PRI_met_phi	The azimuth angle ϕ of the missing transverse energy.
PRI_met_sumet	The total transverse energy in the detector.
PRI_jet_num	The number of jets (integer with value of 0, 1, 2 or 3; possible larger values have been capped at 3).
PRI_jet_leading_pt	The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the leading jet, that is the jet with largest transverse momentum (undefined if PRI jet num = 0).
PRI_jet_leading_eta	The pseudorapidity η of the leading jet (undefined if PRI jet num = 0).
PRI_jet	leading phi The azimuth angle ϕ of the leading jet (undefined if PRI jet num = 0).
PRI_jet_subleading_pt	The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the leading jet, that is, the jet with second largest transverse momentum (undefined if PRI jet num ≤ 1).
PRI_jet_subleading_eta	The pseudorapidity η of the subleading jet (undefined if PRI jet num ≤ 1).
PRI_jet_subleading_phi	The azimuth angle ϕ of the subleading jet (undefined if PRI jet num ≤ 1).
PRI_jet_all_pt	The scalar sum of the transverse momentum of all the jets of the events.

In addition to these 30 features, there are three more attributes to the data, *EventId*, *Weight* and *Label*. They are not used as feature attributes, and are thus removed from the dataset prior to datahandling and analysis.

The features are categorized by a **PRI** or **DER** handle. **PRI** stands for primitives, they are "raw" quantities about the bunch collision from the detector, all related to the momenta of the particles. **DER** stands for derived, they are computed quantities based on the primitive features. It is also worth noting that

- Variables are floats unless specified
- All azimuthal ϕ angles are in radians, in the $[-\pi, \pi]$ range
- Energy, mass and momentum are all in GeV
- All other variables are unitless
- Variables put to -999.0 are either meaningless or cannot be computed. In our case we put them to np.NaN.
- The mass of the particles have not been provided to the dataset, and can thus be neglected for this project