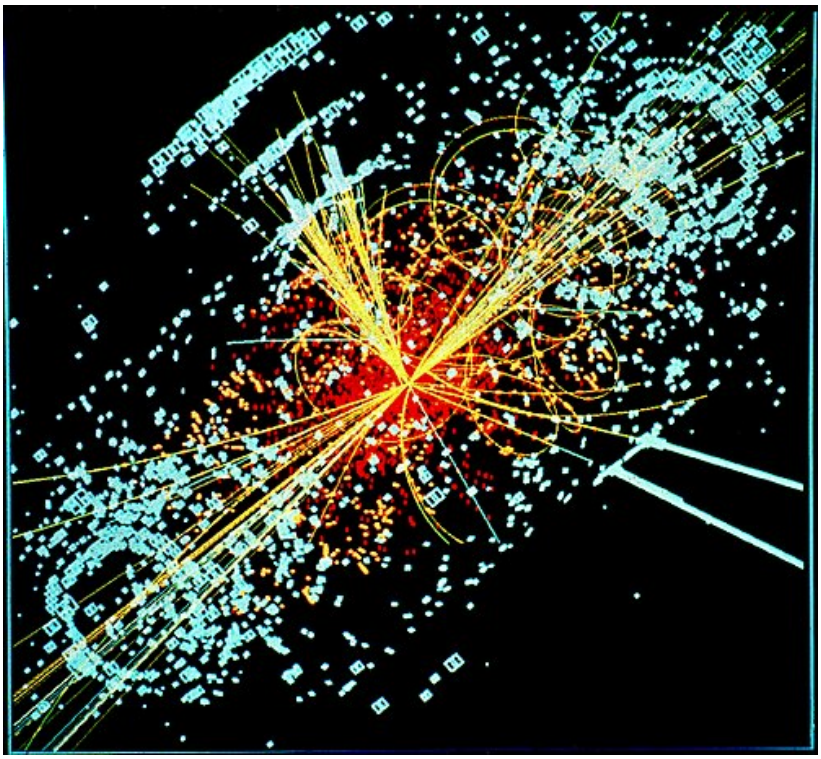
Searching for the Higgs through supervised and unsupervised machine learning algorithms

Comparing different machine learning algorithms in their ability to separate signal from background

Sakarias Frette & William Hirst

Spring 2022



Source, accessed 23.03.22

University of Oslo

Abstract

In this rapport we deployed both supervised and unsupervised machine learning methods to search for the Higgs boson in the Higgs challenge data set. We compared the methods performance in their ability to separate the signal from background with the metrics of area under the ROC curve, AMS and separation of mean. We found that the supervised methods all performed better than the unsupervised methods. The top performer was XGBoost with an AMS score of 3.56 and an area under the curve of 0.92. Although we found the supervised methods greatly outperformed the unsupervised, the addition of alternative metrics better suited to highlight the advantages of all methods (like bias and interpretability) could lead to more competitive results.

I. INTRODUCTION

The Higgs boson was the "final" piece of the Standard Model, and was experimentally confirmed at the LHC in 2012. Through spontaneous symmetry breaking it is responsible for giving mass to fermions and to massive bosons. It's discovery in 2012 showed a 5.9 sigma[1] significance, concluding with very 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. It generates by itself approximately 1 petabyte of raw data every second from proton proton collisions. The amount of data to analyze will increase much further with Run 3[1] and Run 4 over the next decade, thus taking advantage of numerical methods like machine learning is pivotal if scientific development is to keep up with technological development.

Although the standard model is very successful, there are still many problems yet to be solved, such as the hierarchy problem, dark matter and gravity. Many candidates are suggested, from supersymmetry to new vector gauge bosons. Regardless of the theory, we need efficient data analysis tools to help us. One of the most popular choices of analysis tools is the use of machine learning. Machine learning has already been used in many aspects of particle physics, including the discovery of the Higgs boson. Thus far only supervised machine learning has been applied, but in the future many hope that the generality of unsupervised methods can be exploited in the search for new physics.

In this report we compare the efficiency of multiple machine learning methods and data analysis tools to discover Higgs bosons in ATLAS dataset, a competion that was run in 2014[2]. Specifically we will compare supervised to unsupervised machine learning methods in their ability to separate signal from background, where signal is the higgs and background is the standard model events.

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 m. In the implementation section we discuss data handling, the software of choice as well as hyper parameter searches for optimal hyper parameters for each method. In the results and discussion section we present all results as well as discuss any notable observations. In the final section, conclusion, we summarize all notable results and observations of interest.

II. THEORY

A. Higgs boson decay

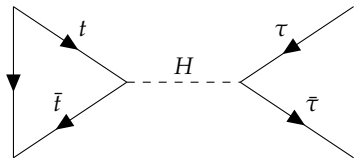


Figure 1: Feynman diagram for dilepton final state for the decay of $H \to \tau\bar{\tau}$.

The Higgs boson has a short lifespan of about $1.6 \cdot 10^{-22}s$, which leaves no hope for any detector of measuring it. In fact, today's detectors can detect everything with a lifespan 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.

[1] At ATLAS, each major data gathering session is called a run. Run 3 will probably begin summer of 2022, ref LisHep-2015

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

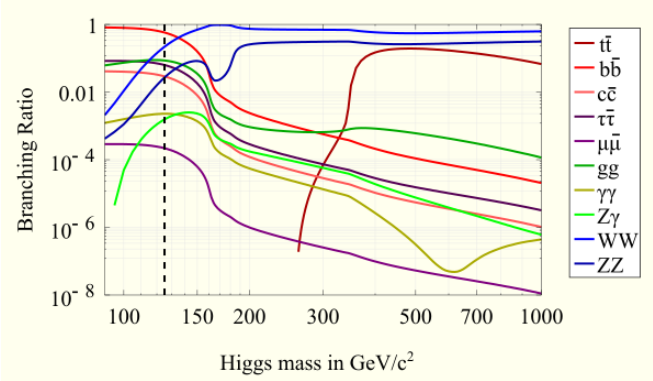


Figure 2: Branching ratio of Higgs decay as function of Higgs mass. Source, accessed 23.03.22

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 $H \to b\bar{b}$ decay is the most occurring decay. At higher energies we see that decay into vector bosons takes over, with only $t\bar{t}$ as a significant fraction representing the decay into fermions.

In this challenge we aim to detect the decay of $H \to \tau\bar{\tau}$, where the higgs was produced by proton-proton collision. In figure 1 we have plotted the simple Feynman diagram of the Higgs into $\tau\bar{\tau}$, where the higgs was created in a top-quark loop produced by proton-proton collision. Given the mass of the higgs of around 125.1 GeV, several standard model background processes will mimic the signal. In the next we will discuss the ones relevant for this rapport.

B. The background

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 diagrams of the Z- and W-decay are shown in figure 3.

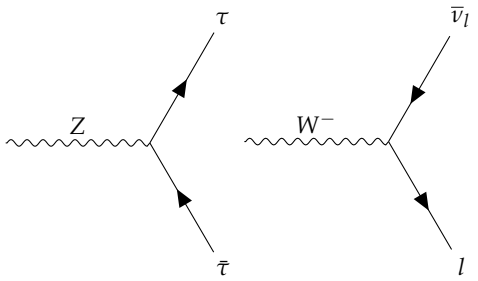


Figure 3: Feynman diagram for the Z-boson decay with two τ-leptons and W-boson decay to charged and neutral lepton.

The third and final background process is the hadronic decay of a pair of top-quarks. The top-quark of the pair decays into a bottom-quark and a hadron mediated by the W^+ boson. The same particles are produced by the anti top-quark, only with an anti b-quark and a W^- boson. Half of the third process is shown in figure 4.

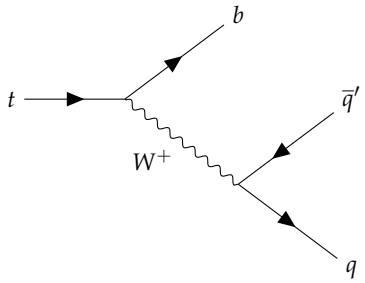


Figure 4: Feynman diagram for the top quark decay with 3 quark final state

C. Supervised Machine learning

Most applications of machine learning in physics today use supervised machine learning methods. What makes supervised machine learning different from unsupervised, is the exposing of target labels during training. This means that all supervised classifiers will be specified to separate one specific target.
The in depth theory of the different supervised models will not covered in this section, however there will be a summary of the different methods. For an in depth explanation 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 work in a rapport from 2021[3].

1. *Feed forward neural networks*

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 5. 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.

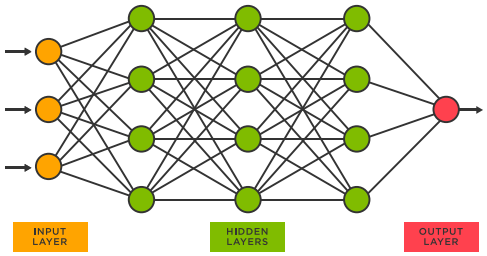


Figure 5: An illustrative image of dens feed forward neural network. Source, accessed 22.03.22

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

In the regression case a decision tree divides the predictor space (the set of possible values $x_1, x_2, \ldots, x_p$) into J distinct and non-overlapping regions $R_1, R_2, \ldots, R_J$. 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

$$R_1 : \{X | x_j < s\}, \tag{1}$$
$$R_2 : \{X | x_j \geq s\}, \tag{2}$$

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

$$\text{MSE}(j, s) = \sum_{i:x_i \in R_1} (y_i - \bar{y}_{R1})^2 + \sum_{i:x_i \in R_2} (y_i - \bar{y}_{R2})^2 \tag{3}$$

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 $p \times N$. This shows that we can actually afford 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 region 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:

$$\tilde{\mathcal{C}}(\mathcal{F}_m(x_i), y_i) = h_m(x_i), \tag{4}$$

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

$$\mathcal{F}_{m+1}(x_i) = \mathcal{F}_m + h_m(x_i). \tag{5}$$

The XGBoost [4] framework used in this analysis enables a gradient-boosted algorithm using decision trees as the weak classifiers. It was initially created for the Higgs ML challenge. Since the challenge, XGBoost has become a favorite for many in the machine learning community and has later won many other machine learning 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

Given that in the case of unsupervised machine learning, we do not expose the classifiers to any labels during training, the hope is that this methods will be more general than in the case of supervised. The idea is that the methods can detect anomalies in the data and will classify this anomalies as signal. In this section we will explain the two methods used in this rapport.

1. *Anomaly detection*

In this section we will introduce the concept of anomaly detection. Contrary to supervised machine learning which trains to detect one specific signal, the concept of anomaly detection is to detect any data which deviates from the "norm". Using supervised methods can be very effective, but is at the same time very model dependant. Anomaly detection aims to find signals, even if one is not sure what one is looking for.

Suppose we were looking for new physics, such as new vector bosons like Z′ or $W^{\pm\prime}$[3], there are so many model candidates to try. With supervised learning you would have to choose one specific candidate and train your classifier to detect it. In the case of anomaly detection, we do not have to choose or assume one specific theory. Instead we simply train a classifier to detect any deviation from the data it is given, in our case this would be the standard model background.

In this report we are looking for the Higgs boson. As explained in the previous sections, we have binary classification, thus we have the standard model background events which will be our training data, and the Higgs events which will be our anomalies.

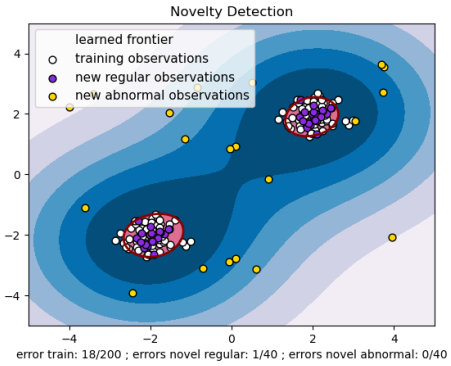


Figure 6: An illustrative image of anomaly detection using a one class support vector machine. Source, accessed 23.03.22

[3] More on candidates for possible CP restoration can be found here G. Senjanovic and R. N. Mohapatra and G. Altarelli, B. Mele, M. Ruiz-Altaba .

In figure 6 we see an example of anomaly detection, where a classifier separates 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.

2. *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. The goal is that by compressing the data into a much smaller latent space, we force the autoencoder to pick up on the most important features and trends. An illustration of an autoencoder is shown in figure 7.

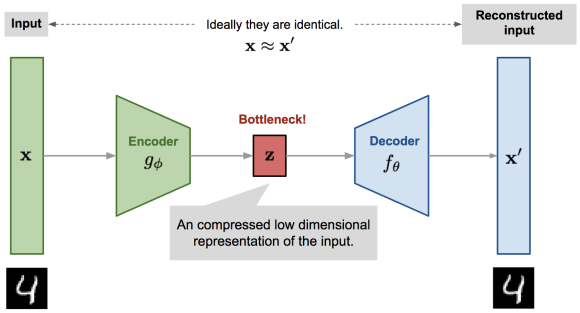


Figure 7: Diagram for auto encoder architecture, using image reconstruction as example. Source, accessed 23.03.22

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 sent in, reducing the number of nodes for each layer until it reaches the latent space, depicted in figure 7 as $\mathbf{z}$. 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 $\mathbf{x'}$. The deconstructed information is given as

$$\mathbf{z} = g_\phi(\mathbf{x}), \tag{6}$$

and reconstructed information is given as

$$\mathbf{x'} = f_\theta(g_\phi(\mathbf{x})), \tag{7}$$

where the parameters (θ, ϕ) are parameters tuned during training to best reconstruct the data. In other words the goal is to choose the variables such that

$$\mathbf{x} \approx f_\theta(g_\phi(\mathbf{x})). \tag{8}$$

To find these ideal parameters we hope to minimize the mean squared error, which would in this case be defined as

$$L_{AE}(\theta, \phi) = \frac{1}{N} \sum_{i=0}^{N-1} \left(\mathbf{x}^i - f_\theta(g_\phi(\mathbf{x}^i)) \right)^2. \quad (9)$$

The use of this method in this project will be semi-unsupervised learning, because we do not show the neural network the signal labels, but rather the data it should aim to reproduce. From here on out we will however describe the auto encoder as an unsupervised method for simplicity.

Given adequate training, the hope is that the auto encoder would be able to (relatively) accurately reconstruct any background, but struggle to reconstruct signal. In this way we can use the reconstruction error to separate the signal from the background.

3. 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.

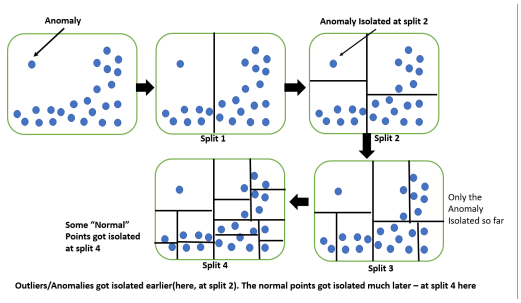


Figure 8: Cuts done to separate the outlier from the inliers. Source, accessed 26.04.22

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 on how many splits do you need to completely isolate the data. In figure 8 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

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}. \quad (10)$$

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

$$c(n) = 2H(n-1) - (2(n-1)/n), \quad (11)$$

where $H(i)$ is the harmonic number, and n is the number of instances in the data set. Because s is monotonic to $h(x)$, i.e $E(h(x)) \to 0, s \to 1$ and $E(h(x)) \to n-1, s \to 0$, 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

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.

1. 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 the rate of true positives 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 9.

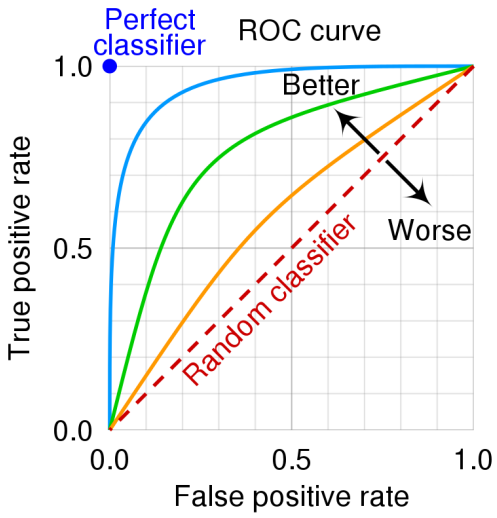


Figure 9: An illustrative image of a ROC curve. Source, accessed 19.03.22

In the ROC-curve figures used in this rapport 4 curves were used; class 0, class 1, micro average and macro average. Class 0 and 1 are simply the true positives for the two classes 1 and 0. As for micro and macro average they are the average score taken for class 1 and 0. The difference between them are when the average is taken. In the case of macro we calculate the average for each class individually, whereas in the case of micro the average is taken for all classes at the same time. This can lead to small deviations between the two curves. In the comparison between different methods, the class 1 values will be used.

2. Distribution histograms

As was previously mentioned in II E, 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. We will aim to highlight any separation in the data by studying the distance between the mean of the output for the signal and background. As a final perk, it will additionally give us some insight into the the ideal values for the machine learning output threshold, which defines the signal.

3. AMS

When on the search for new physics, there are two results of importance, discovery and exclusion. In the case of discovery one often uses significance as a metric, i.e the significance of a discovered amount of potential signal. In this rapport we will define the significance using AMS.

We begin 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 amount of signal events defined by the classifier. 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 distribution given as

$$P\left(n \mid \mu_{\mathrm{s}}, \mu_{\mathrm{b}}\right)=\frac{\left(\mu_{\mathrm{s}}+\mu_{\mathrm{b}}\right)^{n}}{n!} e^{-\left(\mu_{\mathrm{s}}+\mu_{\mathrm{b}}\right)}, \tag{12}$$

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 $\mu_s = 0$ and $\mu_s > 0$.

$$\lambda=\frac{P\left(n \mid 0, \mu_{\mathrm{b}}\right)}{P\left(n \mid \mu_{\mathrm{s}}, \mu_{\mathrm{b}}\right)}=\left(\frac{\mu_{\mathrm{b}}}{\mu_{\mathrm{s}}+\mu_{\mathrm{b}}}\right)^{n} e^{\mu_{\mathrm{s}}}=\left(\frac{\mu_{\mathrm{b}}}{n}\right)^{n} e^{n-\mu_{\mathrm{b}}}, \tag{13}$$

where we have used that $\mu_{\mathrm{s}}=n-\mu_{\mathrm{b}}$.

Further we introduce Willk's theorem[4], which lets us define statistical significance, $\mathcal{Z}$ using λ as

$$\mathcal{Z}=\begin{cases}\sqrt{-2 \ln \lambda} & \text{if } n>\mu_{\mathrm{b}} \\ 0 & \text{otherwise}\end{cases}. \tag{14}$$

By inserting our λ from eq.(13) we find

$$\mathcal{Z}=\sqrt{2\left(n \ln \left(\frac{n}{\mu_{\mathrm{b}}}\right)-n+\mu_{\mathrm{b}}\right)} \tag{15}$$

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

$$\mathrm{AMS}=\sqrt{2\left(\left(s+b+b_{\mathrm{reg}}\right) \ln \left(1+\frac{s}{b+b_{\mathrm{reg}}}\right)-s\right)}. \tag{16}$$

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

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 ratio of s- and b-events can be chosen to fit the preference of the analysis. 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

$$s = \sum_{i \in \mathcal{S} \cap \widehat{\mathcal{G}}} w_i \quad and \quad b = \sum_{i \in \mathcal{B} \cap \widehat{\mathcal{G}}} w_i, \tag{17}$$

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 in this Github repository.

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. In this rapport two data-sets will be considered; training and testing. The training data is labeled and will be used for training all classifiers as well as creating a validation set which will be used for comparison. The test data is not labeled, but was used for a final comparison. After training the classifiers on the training data, the output from the classifiers from the test data was sent to the Kaggle hosts, which calculated the resulting AMS score.

All together the training data consists of 250 000 events, where each event has 30 features[5]. The values of each features varies and can be both binary and continuous. The signal events were generated with a fixed mass of the Higgs of 125 GeV, and the background was generated with the three processes discussed in section II B.

The Monte Carlo simulated dataset is a sparse matrix, due to the choice of certain properties of the system. When one property is present in event a but not event b, the slot for that feature in event b will be filled with a -999.0 value, that is replaced with a NaN value.

In our analysis we have tested several methods of handling our data. These methods are listed as the following:

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

Each function somehow effects the data, and would be applied before any ML method. The *standardScale* scales each feature independently by subtracting the mean and dividing by the standard deviation. $fillWithImputer$ tackles the problem of missing values, but uses a ML algorithm to fill replace the values. *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 the function *removeBadFeatures* is used to remove a feature from all events if that feature has more than a given percentage of "bad" measurements. The last function was tested, but not used for any of the results.

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 methods have been tested with different data handling functions. We then used the data to preform a grid search to obtain optimal parameters and finally we applied the optimized method on the test data.

1. *SciKit-learn, Tensorflow and Keras-Tuner*

Neural networks are highly susceptible to different hyper parameters, and efficient searches for optimal parameters are key when working with high dimensional data. The neural networks used in this report were trained using the Keras-Tuner[6] framework. Keras-Tuner allows neural networks written with Keras in Tensorflow[7] to efficiently do hyper parameter searches, with several search methods. We chose to use Hyperband[8], which is one of the fastest search algorithms.

XGBoost has its own api, where as the two other tree models are fetched from the Scikit-learn[9] api. Both decision tree and XGBoost work well with Scikit-learn infrastructure such as GridsearchCV.

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

2. Neural network

The neural networks are written with a functional structure. Functional structure uses a function call for layers, i.e for layers a, b, then b(a) will connect the two layers. This equals to a sequential link a→b, but allows for more flexible structures.

Since neural networks are essentially a series of matrix multiplications, it cannot deal with NaN values in the dataset. To avoid this problem we used the function $fillWithImputer$ to replace the values. In addition we scaled the data using the function $standardScale$.
Keras-Tuner was used to optimize number of layers as well as number of nodes per layer, learning rate and activation functions.

3. XGBoost and Decision trees

Given that in both decision trees and in the XGBoost method each feature is treated individually, there is no need for scaling. But, where the XGBoost differs from the normal decision trees in that it is not effected by the NaN- values. Therefore the Decision tree algorithm uses the $fillWithImputer$, where the XGBoost does not need to handle this issue.
As for optimization both methods use Scikit-Learn's GridSearchCV[6], which uses a cross validation algorithm. 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.

4. Autoencoders

As was the case for the neural networks, we needed to handle the datapoints as some features had NaN values. Therefore we used the imputer to replace the missing values. Afterwards we scaled the dataset with standard scaling.

For node optimization we set varying intervals on each layer, 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 was also tuned, and the optimizer used was ADAM.

[6] More on GridsearchCV api can be found here.

5. Isolation forest

Due to the underlying algorithm of the method, no tuning was possible for the paremeters. Instead the cluster algorithm is used with the standard hyperparameters set by Scikit-Learn.

The algorithm does require non-NaN values, so that it can make the necessary splits for the features. Therefore we used the $fillWithImputer()$ function to fill substitute all NaN-values. We did not perform any scaling of the data before clustering.

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 distribution of values for a sample of features are plotted in figure 10. It is obvious from the figure that there are some features that exhibit clear trends and some that do not. As is 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.

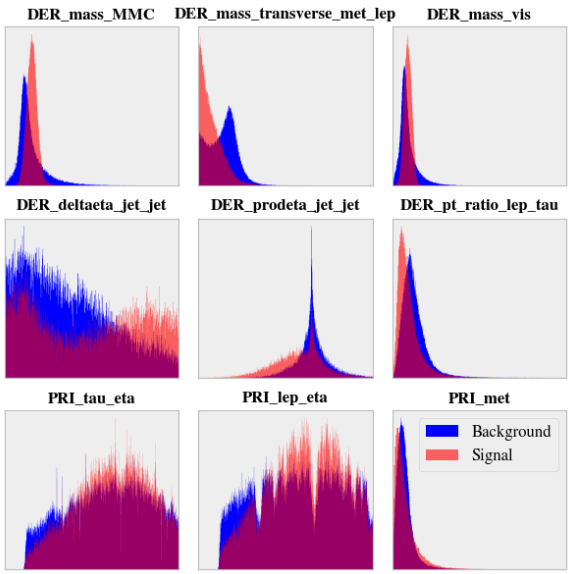


Figure 10: A visualisation of the spread of values for 9 given features (see appendix). Blue representing background and red representing signal.

To further inspect the difference in value distribution of the different features we made a boxplot presented

in figure 11. The figure displays several features where the signal and the background equal in both median and distribution. But, as was evident from figure 10, 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 differences in the 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.

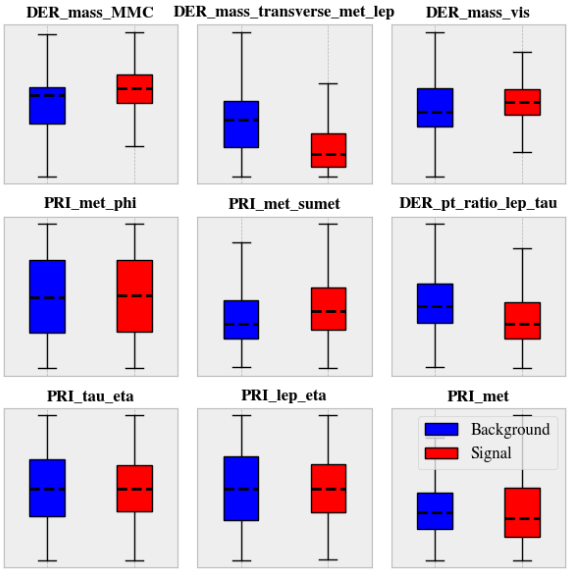


Figure 11: A boxplot for 9 given features, comparing background (blue) signal (red).

B. Hyperparameter search

In this section we will present the ideal parameters that define the structures of the different machine learning algorithms.

1. Neural network

To fine the optimal parameters for the neural network we used the keras Tuner. 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 tuning, 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 figure 12.

Each dense layer in the network uses a leakyRelu activation function with a threshold of 0.1. The ideal learning rate was found to be 0.0015.

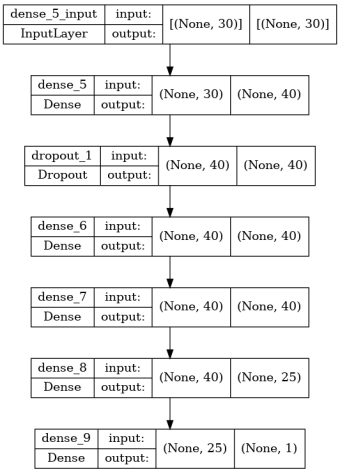


Figure 12: Neural network layout after hyper parameter search using Keras-Tuner.

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 [10] for all but 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

Using the Keras Tuner, we found the that the ideal number of hidden layers for the encoder and decoder were the same amount, 3. The ideal activation function the encoder was Tanh for the first layer and leakyReLU for the other layers, and for the decoder the second layer had Tanh, whilst the remaining layers had LeakyReLU. The α parameter was set to 0.1. Additionally we found the optimal learning rate to be 0.0015, and in figure 13 we present the ideal number of nodes in each layer.

C. Performance

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.

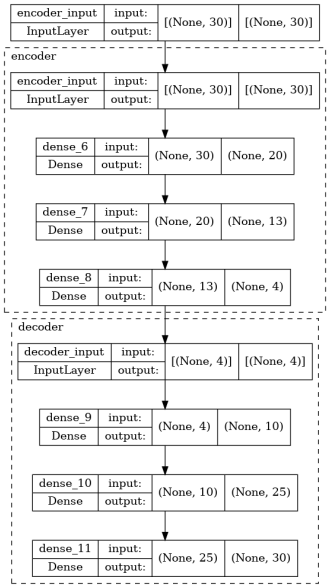


Figure 13: Auto Encoder layout after hyper parameter search with Keras-Tuner.

1. Neural network

In figure 14 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.

In figure 15 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 lower output values and the signal has likewise been separated to higher values. The extent of the separation is made evident through the differences in mean value, 0.47.

A further interesting point is made by a study of the median values of both the signal and background. The difference in median is 0.98. 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.

In figure 16 we plot resulting ROC-curve for the output. From the figure we observe that the model has greatly curved towards an optimal model ([0,1]). The curve creates an areal of 0.92, which shows sign of a good classifier, but with room for improvement.

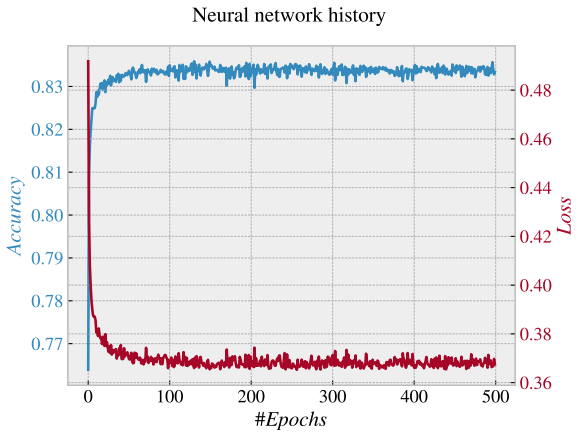


Figure 14: A plot of the training history for the neural network over 100 epochs. All parameters were for the network were decided in the grid search from IV B 1

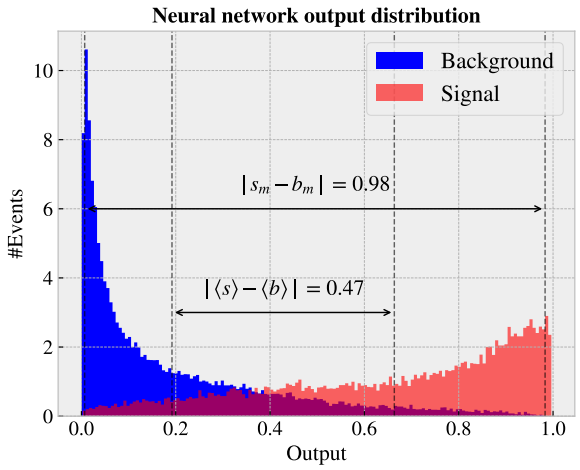


Figure 15: A plot of the output distribution for the neural network after 500 epochs. The distance between the mean and median for s and b is visualized in the center of the plot.

2. Decision trees

In figure 17 we plotted the output distribution of the decision tree. Immediately we observe a far more discrete 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 a single 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 than for the neural network. The difference in median was 0.85 and the difference in mean was 0.40. This indicates that the decision

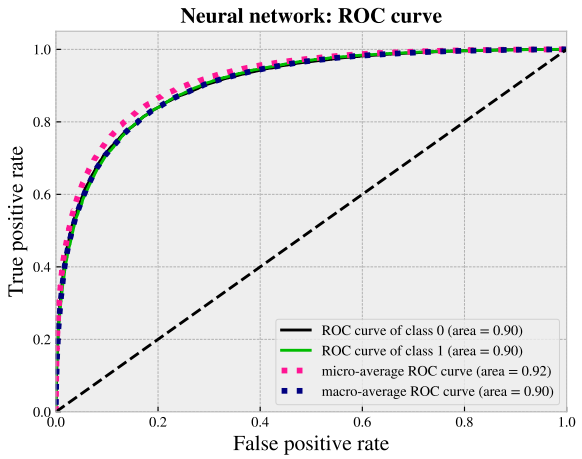


Figure 16: A plot of the ROC-curve for the neural network after 500 epochs.

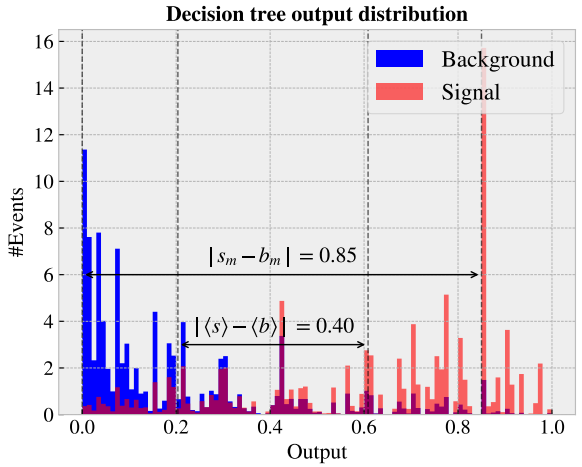


Figure 17: A plot of the output distribution for the decision tree algorithm. The distance between the mean and median for s and b is visualized in the center of the plot.

tree preformed worse than the neural network.
In figure 18 we plotted the ROC-curve from the output of the decision tree. Again we observe that the decision tree preformed worse than the neural network, only achieving an area under the curve of 0.87.

3. XGBoost

In figure 19 we plotted the output distribution of XGBoost. The figure is rather similar to the one produced by the neural network. The main difference is in the location of the peak of the signal, which is now skewed further from 1. This results in the decrease in distance of median to 0.92. On the other hand the XGBoost has increased the distance in mean to 0.50, thereby outperforming the neural network ever so slightly by 0.03. 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 values.

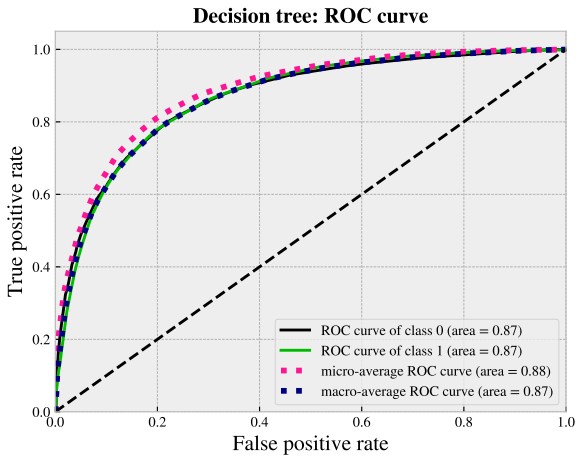


Figure 18: A plot of the ROC-curve for the decision tree algorithm.

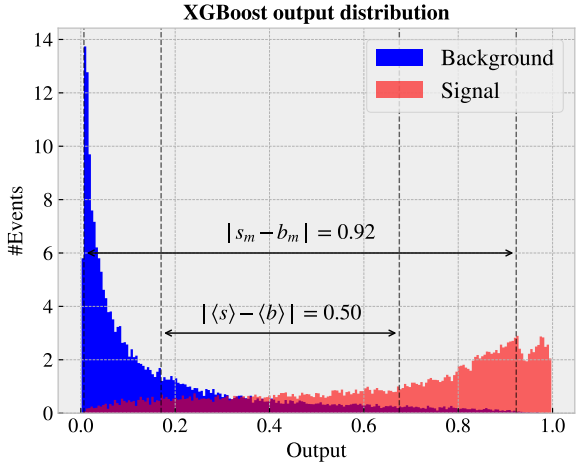


Figure 19: A plot of the output distribution for the XGBoost model. The distance between the mean and median for s and b is visualized in the center of the plot.

It is interesting to note that the mean for the background in figure 19 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, 19 and 15 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 signal is unrealistically high, the neural network would have an advantage. In the AMS score (to be discussed in section IV D), 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

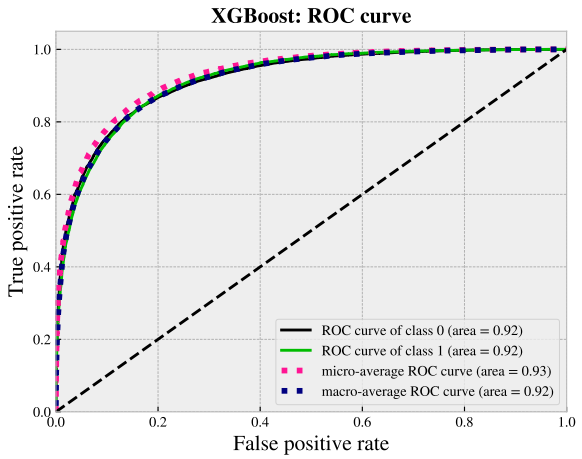


Figure 20: A plot of the ROC-curve for the XGBoost.

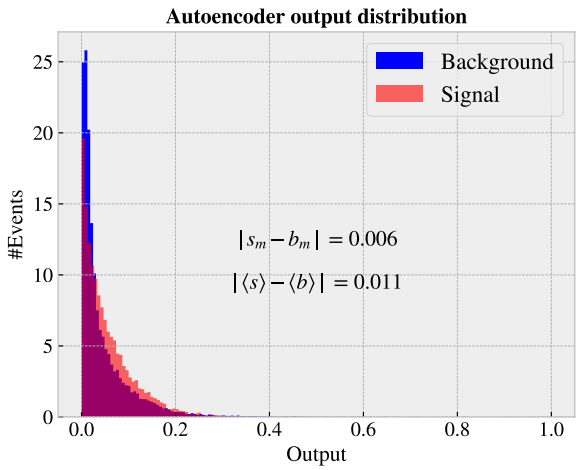


Figure 21: A plot of the output distribution for the Autoencoder model after 50 epochs.

dominant when using this metric. Finally we study the ROC-curve of the XGBoost, figure 20. The figure displays that the XGBoost better classifies the data than both the neural network and the decision tree, achieving an area under the curve of 0.92. Thereby beating the neural network and the decision tree by 0.02 and 0.05 respectively.

4. *Autoencoder*

In figure 21 we see the output of the autoencoder. In the figure we observe a far worse separation between the signal and background. The two are practically on top of each other, meaning that the autoencoder has struggled to find any trend in the data which allows it to separate signal from background.

In figure 22 we plotted the ROC curve created by the output. In the figure we observe a far worse curve than the supervised method, only achieving an area under the curve of 0.58. Although this is not very impressing, it is slightly better than guessing. Given the poor results, we wanted to try and improve by using the *removeOutliers*(). The idea was to plot the distribution of the auto encoder, but with all outliers removed, i.e output larger than 5 times the standard deviation of the background.

In figure 23 we plotted the result after removing the outliers. By comparing figure 21 and 23, we see that by removing outliers the autoencoder has increased the separation of the mean by more than 3 times, achieving a separation of 0.034. In figure 24, we plotted the resulting ROC-curve. In the figure we observe that although the separation of mean has improved, the area on the curve has not.

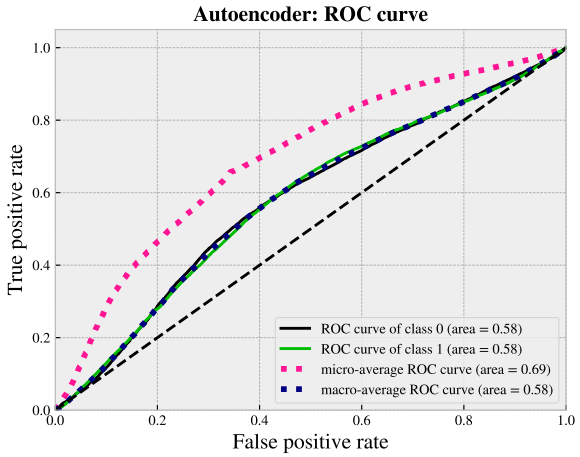


Figure 22: A plot of the ROC-curve for the Autoencoder model.

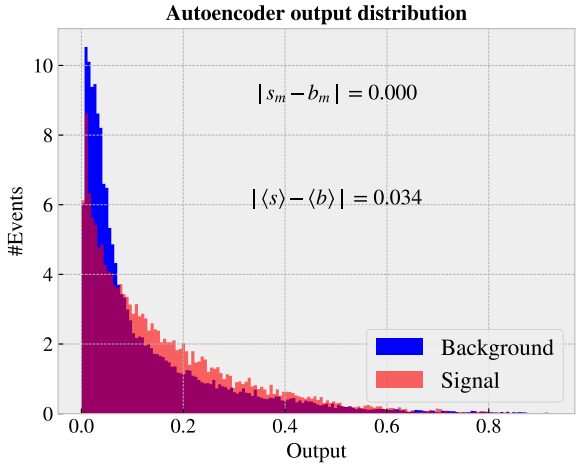


Figure 23: A plot of the output distribution for the Autoencoder model after 50 epochs, having removed outliers of 5 sigma.

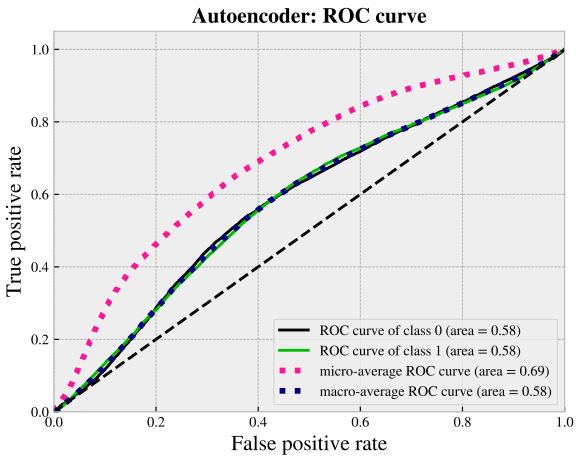


Figure 24: A plot of the ROC-curve for the Autoencoder model.

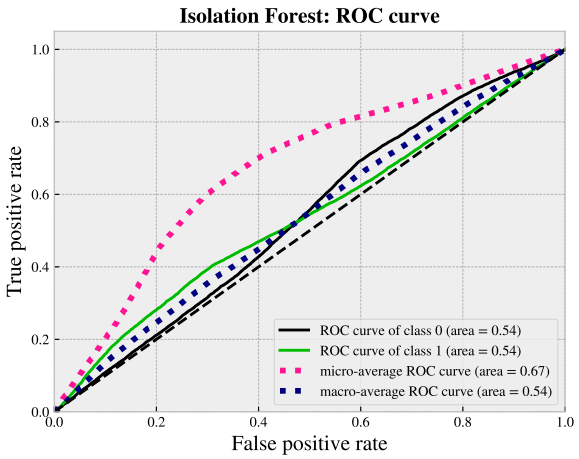


Figure 26: A plot of the ROC-curve for the Isolation forest model.

5. *Isolation forest*

In figure 25 we have plotted the distribution of the output for the isolation forest. In the figure we see practically no separation between the signal and the background, almost resembling the output from the auto encoder. This is further evident from the ROC curve 26, which shows and area under the curve of 0.54.

One could suggest other sets of hyper parameters, but we cannot do any tuning to claim it a truly unsupervised method. Contrary to the auto encoder, there is no way of tuning the parameters of the isolation forest without exposing it to the labels of the training data. There is also the possibility that some handling of the data could improve the score of the isolation forest, but sadly we did not have time to test this.

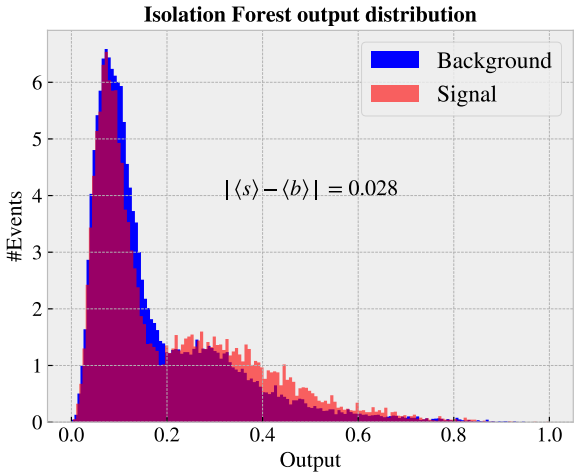


Figure 25: A plot of the output distribution for the Isolation forest model.

D. Comparison

In this section we will compare the results on both the training data and the test data. Firstly a comparison is made between the supervised and unsupervised methods separately, then finally a comparison is made between all method. In this section you will also present the AMS results for all the methods in table II.

1. *Supervised methods*

By studying the results on the training data we saw that in both separation of mean and in ROC-curve area the XGBoost method out preformed the decision tree and the neural network. Additionally the XGBoost also achieved the highest AMS score of 3.56, beating the scores of the neural network, 3.30 and the decision tree 2.83.

Additionally to their performance on the data, there are other advantages and disadvantages between the methods when used in high energy physics. Two of the most important attributes to discuss are interpretability and bias.

In any scientific analysis every result has to be accounted for. Therefore interpretability can be an important factor when choosing a ML method. Simple decision trees are by far the easiest to interpret and dissect. Every cut can be visualized, which makes the method highly transparent. As for XGBoost, we deploy several, sometimes hundreds of trees. For every tree we add the hope of visualizing cuts and understanding results diminishes. What boosted decision trees gain in results, it looses in interpretability. Neural network are by far the hardest to interpret, and are often called black boxes. Especially in the case of deep learning, where the number of tuneable parameters are incredibly high.

Similarly to interpretability, bias is crucial in any scientific research, and is ideally as small as possible. In the case of bias the decision trees and the XGBoost method are fairly similar. For neural network this is not always the case. We will discuss why in section IV D 3.

2. Unsupervised methods

In the case of 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 achieved an AMS score 3 times larger, as shown in table II. 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 was framed as a reconstruction problem, you can tune the algorithm without prior 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 better tuned 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 with Tensorflow, is designed down to almost every last detail, with full oversight of the layers, nodes, activation functions etc.

Finally there is the question of unsupervised and semi-unsupervised. As is briefly mentioned in previous sections, the auto encoder is in theory more semi-unsupervised than unsupervised. This is due to the tuning of parameters on the background data. Where the isolation forest is left completely unbiased, the auto encoder has been exposed to a label, i.e the background. This means that the isolation forest has better potential to be a completely general classifier.

3. Collective comparison

In the supervised comparison we mentioned the importance of bias in the different machine learning algorithms. Bias can come in many ways, one of which is the data handling.

Decision trees, neural networks, auto encoders and isolation forest all have one thing in common. They can not handle NaN values. How one removes/replace these values can sometimes add bias to the analysis. In this rapport we have simple used the imputer method to replace the NaN-values. One of the main issues with doing so is that in some cases we could risk breaking the laws of physics. For instance; given a poor replacement of the momentum in any event, we could break the law of conservation of momentum. 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 this problem and it should be kept in mind when comparing the different machine learning methods.

A possible solution to the sparse data problem could be to change some of the features, by restructuring the information. For example in the case you have three features that hold the transverse momentum of the first, second and third lepton. In the events that you only measure two leptons, there would naturally be a hole in the data. We could solve this by rewriting the three features as two features. The first being the number of leptons and the second being the total momentum of all lepton. This would successfully remove all NaN values, but at the same time you would loose some information. The restructuring of the data can in this way introduce bias to any method.

This very problem is yet to be solved by the high energy physics community and is one of the reasons that physicists often prefer the XGBoost framework. XGBoost can as mentioned earlier handle missing data, and thus can much easier preserve the laws of physics.

We can now move on to compare the different results for each methods. In table I we have displayed the mean separation and area under the curve for the output from each method. In the table we observe that the supervised methods were all far better at separating the signal from the background. The dominance of the supervised methods can be explained by the fact that they are all trained specifically to separate Higgs from the background. In the case of unsupervised methods they are instead trained to detect deviations.

Table I: Output separation and ROC score for all methods

Method	Mean separation	ROC score
Neural network	0.47	0.90
XGboost	0.50	0.92
Decision trees	0.46	0.87
Autoencoder	0.01(0.03)	0.58
Isolation forest	0.028	0.54

In table II we have presented the AMS score achieved by all of the methods. As mentioned previously, the AMS score was calculated on a separate data set, testing which did not have labels. In this table we again observe that the supervised methods all achieve a much

higher score than the unsupervised methods. The worst supervised method achieved an AMS score almost three times larger than the best unsupervised method.

Table II: 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

Another point of interest is the usage of the weights of each event. In the case of the supervised methods, the weights of the events could have been used in initialization of weights in the classifiers. When researching different attempts at the ML Higgs challenge, we found that most, if not all the top contenders used the event weights in the initialization. As mentioned this is only relevant in the case of supervised learning, given that the unsupervised methods would not have access to the weights of the signal. Therefore, given that one of the purposes of this rapport is the comparison between supervised and unsupervised we have chosen not to do this.

Lastly we must be aware that the search in this project is somewhat tailored to supervised methods. Just because the unsupervised methods performed worse than the supervised methods in the search for Higgs here, does not mean that they are poor classifiers. Model dependent algorithms are highly accurate, but limited in their scope. Model independent algorithms are less accurate, but are able to widen the scope of the search. The metrics used in this report do not facilitate the possibility for the unsupervised models to be tested for this. It would therefore be interesting to use different metrics to compare the methods in the future. One alternative could be to test pre-trained supervised and a unsupervised algorithms on a data set with a similar but different signal. In doing so we might see results slightly different for what we found in this rapport.

V. CONCLUSION

In this report we aimed to compare the efficiency of several machine learning algorithms in the use for searching for new physics. Both supervised and unsupervised methods were deployed on the Higgs challenge[7] dataset, using Tensorflow and SciKit-learn Api's. We found that all the supervised methods outperformed the unsupervised methods both in comparing ROC scores and AMS scores.

We also found that among the three supervised methods, XGBoost performed slightly better both feed forward neural network and ordinary decision trees in AMS, 3.56 as well as area under the ROC score, 0.92 and mean separation 0.50. It is not however trivial that XGBoost will always be better than a neural network when doing a model dependent search. For the case of the unsupervised methods, it was shown that, despite having lower area under the ROC curve, the Isolation forest model achieved better AMS score, three times the AMS score of the auto encoder.

Although the metrics we use are well defined, the evaluation of the models must be read with an asterix. Some algorithms require padding and scaling, and some do not, which leads to a slightly unfair comparison. Additionally it is worth noting the fact that the metrics chosen in this rapport are, although not purposely designed to highlight the advantages of supervised learning. Defining alternative metrics better suited to highlight all advantages (for example interpretability and bias) would be interesting to pursue in the future.

[7] Information surrounding the Higgs challenge can be found on there website, here.

REFERENCES

1. Aad, G. *et al.* Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Phys. Lett. B* **716,** 1–29. arXiv: 1207.7214 [hep-ex] (2012).
2. Frette, S., Hirst, W. & Jensen, M. M. A computational analysis of a dense feed forward neural network for regression and classification type problems in comparison to regression methods, 5. https://github.com/Gadangadang/Fys-Stk4155/blob/main/Project%202/article/Project_2_current.pdf (2022).
3. Frette, S., Hirst, W. & Jensen, M. M. Bias variance analysis, 3. https://github.com/Gadangadang/Fys-Stk4155/blob/main/Project%203/article/Project_3_current_part2.pdf (2022).
4. The XGBoost Contributors. *XGBoost* version 1.5.2. https://xgboost.readthedocs.io/en/stable/#.
5. Liu, F. T., Ting, K. M. & Zhou, Z.-H. *Isolation Forest* in *2008 Eighth IEEE International Conference on Data Mining* (2008), 413–422.
6. O'Malley, T. *et al. KerasTuner* https://github.com/keras-team/keras-tuner. 2019.
7. Martín Abadi *et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems* Software available from tensorflow.org. 2015. https://www.tensorflow.org/.
8. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. & Talwalkar, A. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research* **18,** 1–52. http://jmlr.org/papers/v18/16-558.html (2018).
9. Pedregosa, F. *et al.* Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **12,** 2825–2830 (2011).
10. Scikit-learn developers. *DecisionTreeRegressor* 2022. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html.

Appendix A: Features

Table III: 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