Higgs Challenge

Eneko Ruiz
Marco Wetter

18.08.2019

Contents

1 Preselection and Filtering of Variables

In this section we will discuss which variables we used in our different methods and how and why we further filtered the dataset. The goal is to reduce the amount of variables to increase stability of the methods and reduce the training time.

1.1 Missing Values

At first we considered the data given in the dataset. As we are interested in a decay of a Higgs boson to two leptons, one of which decays hadronic and the other leptonic,

$$\mathrm{H} \rightarrow \tau_{\mathrm{lep}} \tau_{\mathrm{had}}, \tag{1}$$

we filtered the data for missing values. In the given decay we expect at least two jets for our analysis. Otherwise we cannot say whether or not the given decay was present. In the dataset there are events for which some values are missing, e.g. their value is given as -999. If any of the jet related values are missing it indicates that there were too few jets present. Therefore these events were excluded from further analysis.

1.2 Non Discriminative Variables

Upon visual inspection many variables were removed from the analysis. As exclusion criteria we considered the seperability between the signal and background events as well as the general distribution of a variable. If a variable is uniformly distributed over all samples, it will not have any considerable information considering the classification task. An example for a non discriminative variable is given in figure **??**. Here one can see that both background and signal have a very similar distribution and therefore the variable does not increase the separation of signal and background. In figure **??** a uniformly distributed variable is depicted. One such variable also does not yield any information about the process and is therefore also omitted.

1.3 Correlation Matrix

In a final step we considered the linear correlation coefficients of the remaining variables. We omitted variables in which the correlation coefficient as larger than 0.9, as these variables do not add to the seperation of signal and background. In order to reduce the variable space even further such variables were omitted. The final correlation matrix for signal and background is shown in figure **??**. After all these preselections we were left with the following subset of variables:

2 Machine Learning Methods

In this section we will discuss some of the machine learning methods considered for this work. Please note, that our main focus laid on artificial neural networks (ANNs) and boosted decision trees (BDTs).

2.1 Support Vector Machines

The idea of Support Vector Machines (SVM) is to scan the variable space for a few datapoints, that help it seperate the data ideally. Having found such support vectors, the SVM draws a hyperplane in the parameterspace which seperates the data. The issue with this method is that its classification efficiency depends on the seperability of the data. The data obtained from the Atlas experiment are noisy and in no way linear seperable, making this approach only moderately efficient. Nevertheless we implemented a basic SVM and calculated the result on the final data, which is given in section 3.

2.2 Fisher's Discriminant

In order to seperate the data, the Fisher's discriminant tries to find a projection of the data onto a vector which seperates the data as best as possible. The general approach tries to maximize

$$J(v) = \frac{(\mu_1 - \mu_2)^2}{s_1^2 + s_2^2}, \tag{2}$$

where mu_1, mu_2 are the mean values of two different classes after the projection onto the vector, and s_1, s_2 are the standard deviations of the same classes after the projection. This is generally a very efficient approach, even though it suffers a lot from the curse of dimensionality. The higher the dimension of the data, the harder it is to find vectors on which to project. Also the method is inefficient for linear non-seperable data. Therefore this application is assumed to be not optimal for the given application.
We optimized the given example of a Fisher's Discriminant as best as we could, and obtained a result as given in section 3.

2.3 Likeliood

The likelihood classifier given in the template is also called the "naive Bayes estimator". In the training process it tries to estimate the probability density functions (PDFs) of the signal and background data. Afterwards a prediction of a new sample is down by multiplying the signal PDFs and normalizing it with the background PDFs, e.g.

$$y_{\mathcal{L}}(i) = \frac{\mathcal{L}_S(i)}{\mathcal{L}_S(i) + \mathcal{L}_B(i)} \tag{3}$$

The likelihood classifier is generally fast in training, but does not produce good results in testing. We assume that the Likelihood estimator cannot learn the highly dimensional and non-seperable data of the Atlas experiment.

2.4 Boosted Decision Trees

A basic decision tree consists of multiple nodes, each of which splits the data in a special way. The subsample of the data can then be split again with another criterion, until the final stopping criterion is fulfilled. The final nodes are called leaf nodes. In a decision tree a node usually splits the data with the help of a simple criterion, e.g. $var_a > y_{\text{cut}}$. The data is then cut with the help of this criterion and the subsample can be used to define additional cuts. The optimizing criterion for a cut is the optimal separation of signal and background, or the purity of the child nodes. After learning the data, a new sample can be efficiently classified with a decision tree by just running it through the cuts of the node criterion. This makes decision trees fast in evaluation.
Boosted decision trees (BDTs) use an ensemble of many decision trees. The trees are all trained on the data, by reweighting different events. This results in many different trees with different decision boundaries. An event is then classified by making a prediction of the event with every decision tree and summing up the weighted results. This method allows for more stable predictions of decision trees and also enhances the performance significantly.

2.5 XGBoost

XGBoost is a framework which provides gradient boosting for different programming languages. Its basic BDT supposedly gives a score of $AMS = 3.64$ on the higgs challenge. The algorithm uses a gradient dependent algorithm to split the nodes and grow the decision trees. The basic principle the the same as with BDTs.

2.6 Artificial Neural Network

An artificial neural network (ANN) is a collection of perceptrons which are arranged in layers. Every perceptron received a weighted input from all perceptrons of a previous layer. In case of the input neurons the precious layers is the data. The perceptron then gives the sum of all the inputs to the next layer, while usually incorporating additonal weights and activation functions. An ANN with only 2 hidden layers is able to solve many nonlinear problems in machine learning, such as the XOR problem.
Deep neural networks (DNNs), i.e. ANNs with many hidden layers, are widely used in many areas such as speech recognition or image classification. The fine-tuning of those networks is a long and tedious process and often incorporates many task-specific hyperparameters in order to optimize the accuracy of the model. Hyperparameters include the number of layers and the neurons therein,

regularization techniques such as weight decay or dropout, improved optimizers such as the Adam optimizer or stochastic gradient descent and many more. Nevertheless DNNs are able to perform good if used right. Therefore we implemented a neural net with a few hidden layers, to ascertain its performance.
Besides the implementation of ANNs in the TMVA we also implemented a neural net using keras with tensorflow as backend. Keras gives access to more activation functions, layer types and regularization techniques. It also has many different built-in optimizers which are easy to tune.

Table 1: Hyperparameters for the Support Vector Machine

Hyperparameter	value
Gamma	0.1
C	1
Tol	0.01
MaxIter	200
VarTransform	decorrelate, gaussian

3 Results

In this section the results of the above decribed machine learning methods are summarised. In each subsection we give the hyperparameters used for the training and in the end give a comparison of the AMS score of the different methods on the dataset of the higgs challenge.
The score of the XGBoost was obtained in a different program and the AMS score was calculated using the function 'find best ams' given in the template.

3.1 Support Vector Machines

A basic SVM was implemented for reference, with a reduced amount of iterations for the sake of time saving (The SVM was exceptionally slow in both training and evaluation). After a little bit of tuning it performed suprisingly well, with a score close to that of the TMVA MLP. Further optimization of the hyperparameters may have improved the performance a bit, but as we did not expect the SVM to be a good classifier for the given data, we did not improve it further. The final hyperparameters are given in table 1

3.2 Fisher's Discriminant

The Fisher's discriminant did not perform well, even when trying to optimize the results. In fact the performance got worse when we tried to tune the hyperparameters after applying a cut on the data. We therefore left the hyperparameters unchanged and calculated the results. As the hyperparameters are given in the template provided with the data, we do not describe them any further.

3.3 Likelihood

The hyperparameters for the likelihood are given in table 2. Very little effort was put into this, and a more throughout optimization surely would have improved the performance on the final dataset.

3.4 BDT and XGBoost

As BDTs are one of the most used and most promising machine learning methods in particle physics, we tried to optimize the parameters as given in the

Table 2: Hyperparameters of the used likelihood classifier.

Hyperparameter	value
VarTransform	decorrelate, normalise
PDFInterpol	Spline3
NSmoothSig[0]	5
NSmoothBkg[0]	25
NAvtEvtPerBin	10

Table 3: Hyperparameters for Boosted Decision Trees.

Hyperparameter	value
VarTransform	gaussian
NTrees	2000
MinNodeSize	5 %
BoostType	AdaBoost
AdaBoostBeta	0.25
UseBaggedBoost	True
GradBagginFraction	0.3
nCuts	50
UseRandomisedTrees	True
UseNvars	5
NodePurityLimit	0.5
MaxDepth	5

template in order to improve the performance. An important criterion was to try and prevent overfitting, which is a common problem in BDTs. At the same time the BDTs must be able to learn the data, which means that complexity still needs to be rather high. We tried to prune the trees after training, which resulted in a very low AMS. We therefore tried to select the hyperparameters carefully in order to reduce overfitting.

We also implemented a basic BDT in XGBoost in order to compare the results. The hyperparameters were first set according to the follow-up paper of Chen and He, in which they described their approach to the higgs challenge. Afterwards we improved the parameters of the algorithm in order to improve the performance. The tuning was an interplay of recommendations for boosting parameters in general, experimentation on the testdata and bias induced tuning towards a better fit towards the given data, e.g. tuning of the number of trees and iterations so that the dimensions of the data can be projected properly. The resulting hyperparameters are summarized in table 4.

3.5 Multilayer Perceptron

The basic multilayer perceptron given in the template was improved a bit by tuning its hyperparameters. Performance increased for a small increase in complexity, but adding too many layers made the performance worse. This may have many reasons, but was most likely either due to overfitting, e.g. the MLP memorized the training data and therefore could not generalize on the test data,

Table 4: Hyperparameters for XGBoost.

Hyperparameter	value
max depth	6
min child weight	3
eta	0.1
gamma	0.1
objective	binary:logitraw
eval metric	auc, ams@0.15
subsample	0.8
n estimators	500
lambda	1.0
colsample bytree	0.8
alpha	0.1

or because of gradient vanishing, which happens in many deep architectures. In case of gradient vanishing the net cannot learn anymore, because the gradient which is propagated through the net is repeadately multiplied with the activations of the perceptrons. If these activations are smaller than 1 the gradient gets increasingly smaller and the lower layers then cannot update their weights. For the ANN which was created with Keras the observation was almost the same. After adding more than 5 or 6 hidden layers to the net the performance dropped significantly, even after increasing the training epochs. Even techniques such as gradient clipping, weight decay, dropout, gaussian noise and more did not yield any significant improvement in the performance. We also tried to apply a bottleneck to the deep structure, e.g. hidden layers with only few neurons in the middle, and layers with more neurons to the beginning and the end of the net. This also resulted in a poor performance. This is probably due to the fact that bottleneck layers are only useful if there is a basic principle to be learnt from the data. This bottleneck then would force the net to learn this underlying principle, which in return would increase the performance. We hoped that the underlying feature of "Higgs boson produced" could be learnt, but did not find any configuration in which the bottleneck produced any reasonable results.
We then agreed on a smaller net with different activation functions. The size of the hidden layers was chosen such that at first the data is transformed into a higher dimensional space and then is slowly reduced until it is connected to the final layer with only two output neurons. The final hyperparameters with the best performance are given in table 5.

3.6 Results

The tuning of the hyperparameters was done on the smaller subsample delivered with the higgs challenge. After the best configuration was found we downloaded the full dataset and ran the analysis of our methods on it. No further tuning was done after the final results were revealed. A note on the score of the SVM. The laptop used to train the machine learning methods was not able to withstand the high demand that the SVM had. The TMVA crashed while trying to fit it, and therefore no final score was obtained. Nevertheless we multiplied the AMS

Table 5: Hyperparameters for the two types of MLP

Hyperparameter	TMVA MLP	Keras MLP
VarTransform	decorrelate, normalise	decorrelate, normalise, gaussian
Epochs	100	100
Learning rate	0.02	0.001
Number of Layers	2	6
Size of Layers	N+15, N	2*N, 5*N, 2*N, N, N/1.5, N/2.5
Activation Functions	tanh	relu, softplus, softmax
BatchSize	25	25
Errorfunction	Mean-Squared-Error	binary crossentropy
Optimizer	Backprogagation	Adam with default values

Table 6: Final results for the different classifiers.

Classifier	AMS	Transformation
Likelihood	1.458	probability
Fisher's Discriminant	1.469	None
Support Vector Machine	1.519	None
TMVA MLP	2.246	probability
Keras MLP	2.329	rarity
Boosted Decision Tree	2.529	probability
XGBoost		None

received from the small dataset with the average increase in the other methods, in order to obtain a comparable result. The final results of the AMS are given in table **??**.

Table 7: cap1
Hyperparameter

4 Conclusion

We used many different methods and tried to obtain a good AMS for our analysis. In the end the gain was minimal, and an increase to and AMS over 3 seems to be really hard to get by. There are many reasons as to why our analysis did not improve significantly over the initial parameters.
First of all the choice of hyperparameters is crucial. Having never worked with any machine learning method in particle physics, and using many of the methods for the first time on such complex data, we did not have a general understanding of how to approach this task. We consulted multiple references in order to get a good feeling for the hyperparameters. More expertise in this field might have improved the results significantly.
Another point, and maybe the most important one, was the type of methods used. Comparing our basic machine learning methods with the winning teams of the higgs challenge, on can see that in order to obtain a good result the complexity of the method has to increase further. Many teams used XGBoost or DNN together with cross-validation techniques, ensembles and custom boosting functions. Our own implementation of XGBoost did achieve a relatively high score, but didn't even come close to the value obtained by the teams in the challenge. This means that a more detailed implementation of any one machine learning method with custom optimizers, which are designed for the problem at hand, might have yielded a much higher AMS than we obtained in our final analysis.
Finally the selection of the variables is also a huge factor in the performance. We chose to reduce the number of variables in order to reduce the training time and increase the stability. Chen and He did, according to their follow up paper, follow a different approach: in addition to the 30 parameters given in the dataset, they calculated an additional 138 parameters for each datapoint. With the help of these features they improved their performance even turther. Therefore the reduction of the variable space might have been the wrong approach, even though we did not have the computation power to use more variables in our analysis.