Investigating the relative performance of machine learning algorithms

Klaudia Agata Sternik

School of Physics and Astronomy, Queen Mary University of London

Abstract

The relative performance of machine learning algorithms in particle physics analysis has been studied. Boosted decision trees and support vector machines have been investigated through the use of simulated LHC data sets. The average expected performances on unseen data sets have been investigated through the use of the LLR method, together with hyper parameter optimisation. The Jackknife method and Jackknife Variance Estimate were used to find the standard error for the calculated performances. A comparison is presented of the expected performance of these algorithms in rejecting backgrounds from top pair production in searches for di-Higgs signals at the ATLAS experiment at the LHC. After optimising both algorithms, the most important parameters were found to be gamma and cost for SVMs, and maximum tree depth for BDTs. When compared together, SVMs were found superior when a small amount of training data is used, whilst BDTs performed better with more training data. BDTs are more suitable for high energy physics research due to their performance when using large amounts of training data, and lower training times.

Contents

1 Introduction **3**

2 Boosted Decision Trees **3**

3 Support Vector Machines **4**

4 Logarithm Likelihood Ratio **5**

5 Jackknife **5**

6 Results and Discussion **6**

7 Conclusion **13**

8 References **15**

1 Introduction

High energy physics (HEP) has become very reliant on multivariate analysis. The most widely used machine learning algorithms are neural networks and boosted decision trees (BDTs). Outside of the HEP environment, support vector machines (SVMs) are another popular choice for classification problems. In this investigation, an attempt is made at optimising SVMs and BDTs so that they can be analysed at their best performance. Next, the variance of the performance is calculated and analysed to test the algorithms for consistency and overtraining.

The optimal tuning of a machine learning hyper-parameters is dependent on the problem. To ensure realistic results, simulated LHC data sets are used when testing the algorithms. They are tested on their performance in rejecting backgrounds from top pair production in searches for di-Higgs signals at the ATLAS experiment at the LHC when running at 13.5 TeV. The measure of performance is the significance, which is calculated using the Log Likelihood Ratio method. The variance of the significance is then calculated using the Jackknife Variance Estimate.

2 Boosted Decision Trees

Decision Trees are a machine learning algorithm that is commonly used in high energy physics, as well as other research disciplines. [1] They work on a simple cut-based principle, and extend it by continuing to analyse events even if they fail some criteria. This is important as not all events will have every signal or background characteristic, which means too many events would be rejected if classification was only based on a single criterion. Instead however, these events are kept and tested against other criteria in an attempt to identify them as either signal-like or background-like.

[2] The downside of decision trees is their instability. The tree structure is derived from the training sample of data, which can be largely affected by statistical fluctuations. Even a small change in the sample can result in the algorithm deciding to split on one variable instead of another equally viable choice. This alters the final tree structure that can now give a significantly different classifier response.

An effective solution to the shortcomings of individual decision trees increases the statistical stability as well as separation performance. It works by constructing a forest of trees, each derived from the same training sample. The events used are subjected to boosting, which modifies their weights. The boosted decision trees will then collectively classify an event based on

majority vote of each individual tree in the algorithm.

Boosted Decision Trees have a large number of hyper parameters available for customisation, although most wont largely affect the efficiency of the algorithm. Additionally, the Gradient Boost is used in this investigation. GradientBoost is a deviation of the AdaBoost algorithm and it is designed to retain the strong performance of the AdaBoost algorithm while removing the issue of degraded performance in settings with large quantities of noise. This makes it a good candidate for the large collections of data at the LHC. GradientBoost is best used on weak classifiers, so an optimal decision tree would be individuals with a depth of 2 to 4. With this design in place, instances of overtraining are reduced. The exact value of the maximum tree depth (MaxDepth) is to be determined during hyper parameter optimisation together with the number of trees (NTrees). Additionally, the MinNodeSize and nCuts will also be investigated. They are defined as the minimum percentage of training events required in a leaf node, and number of grid points in variable range used in finding an optimal cut in node splitting respectively.

3 Support Vector Machines

During the 1960s, support vector machines were developed for use in pattern recognition problems. Years later it has been improved, spreading it's use to nonlinear separating functions, as well as regression functions. The method has since become a general purpose algorithm, and has been able to compete with other commonly used algorithms in different fields. While it isn't currently popular, SVMs could be applied to HEP research in place of BDTs or Neural Networks if they are found to perform well.

A support vector machine works on the premise of creating a hyperplane that will separate two distinct classes (signal and background), whilst minimising the amount of training vectors (support vectors) used. The support vectors are chosen from the training sample such that they are the closest to the hyperplane. The algorithm will attempt to find the optimal position of the hyperplane at which the margin (distance) between it and the support vectors is maximised. This concept can be extended further to data sets where the two classes overlap, as well as to non-linear problems. This is achieved by a transformation of the variables known as the [3] Kernel trick. These changes do however introduce more hyper-parameters to the algorithm that need to be tuned to optimise the algorithm performance. [4] The most crucial hyper-parameters are Cost and Gamma.

4 Logarithm Likelihood Ratio

[5][6] In order to test the efficiency of algorithms, a good way to measure it is necessary. In a situation where a new signal is being searched for, two hypotheses are set. H_0 states that all detected events are of the background type. H_1 then states that the events are instead a mixture of the background and the desired signal. For a discovery, the hypothesis H_0 needs to be rejected with 5σ significance. When the expected number of signal and background events is s and b respectively, the total observed number of events n follows the poisson distribution. It is possible to find the significance using Likelihood $L(\lambda; x)$ given that it is proportional to $P(n; \lambda)$. The LLR takes the logarithm of the ratio of the likelihoods for both hypotheses. The full equation can be seen below.

$$-2ln\frac{L_{s+b}}{L_b} \tag{1}$$

Other methods of calculating efficiency are available, such as the area under the receiver operator curve (ROC), the use of which is common in HEP.

5 Jackknife

The Jackknife is one of the oldest resampling methods, and was originally proposed by [7] M.H. Quenouille as a method of correcting bias. It was later refined by John Tukey who used it as a way of estimating the variance of an estimator. The second application of the Jackknife is particularly interesting, as it allows the user to calculate the standard error in situations where otherwise it would have been impossible.

Before continuing, it is important to consider the drawbacks to using delete-1 jackknife. It has been shown that the Jackknife estimate of variance tends have an upward bias. There is a method of correcting bias, however due to the uncertainty on the bias estimate, it can make the variance estimate larger. Additionally, it is known to give inconsistent estimates of variance for estimators which are not smooth functions of each data point. This means that the variance estimate doesn't converge to the true value as the sample size increases. It is however possible to choose an estimator that isn't prone to these problems. The jackknife estimate of the standard error does reduce to the commonly used unbiased estimator in the case of the sample mean. Therefore, the estimator used in this investigation is the mean.

[8] Given a random sample, $X = (X_1, X_2, ..., X_n)$, where n is the number

of observations in the sample, a chosen statistic can be calculated as an estimator $\hat{\theta}$ of some parameter θ in the population. In this investigation, the Log-Likelihood ratio is calculated for different data sets to create the sample X. Using the standard Jackknife delete-1 method, n unique Jackknife samples are selected. Each one is constructed by omitting a single data point from the original sample, resulting in samples containing $n-1$ observations each. Those samples can be denoted as $X_{[i]} = (X_1, ..., X_{i-1}, X_{i+1}, ..., X_n)$, where the ith observation is omitted. From each jackknife sample, the estimator of θ is calculated. Those are called the Jackknife replications, they can then be used to estimate the bias and variance of the estimator $\hat{\theta}$ which uses the full sample X. Here, the estimator used is the mean of a sample, which is given as:

$$\hat{\theta}_{(\cdot)} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_{(i)}. \tag{2}$$

The mean of a jackknife sample can be used to get more information about the estimator $\hat{\theta}$. To do this, a pseudovalue $\hat{\theta}_j^*$ is defined as the weighted difference between the value of the estimator for the full sample and the jackknife sample:

$$\hat{\theta}_j^* = n\hat{\theta} - (n-1)\hat{\theta}_j. \tag{3}$$

The mean of the pseudovalues can be written as follows:

$$\hat{\theta}^* = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_{(i)}^*. \tag{4}$$

In this method, the pseudovalues are treated as independent of each other. An estimate of the variance of θ can then be found as the variance of $\hat{\theta}^*$. The Jackknife Variance Estimate is therefore defined as

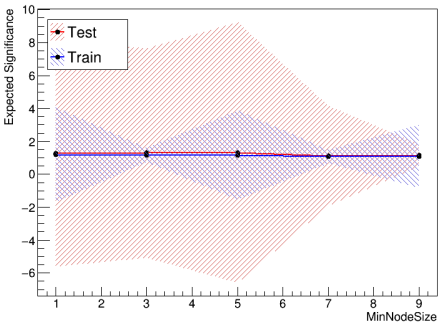
$$Var(\hat{\theta}^*) = \frac{\sum_{j=1}^{n}(\hat{\theta}_{(j)}^* - \hat{\theta}^*)^2}{(n-1)n},$$

The Jackknife Standard Error can then be written as

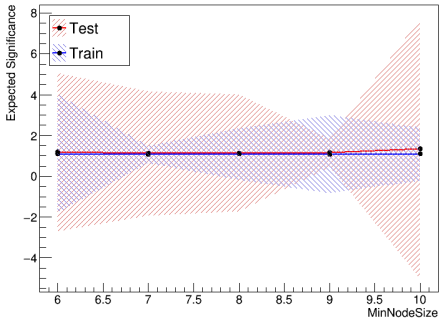
$$SE_J = \sqrt{Var(\hat{\theta}^*)} \tag{5}$$

6 Results and Discussion

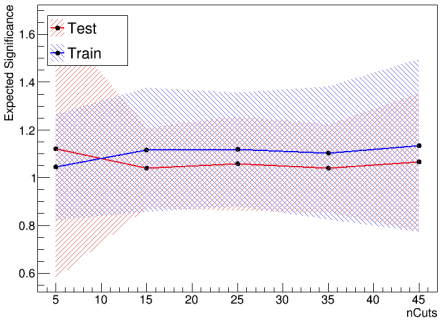
The investigated hyper parameters of Boosted Decision Trees were maximum tree depth, number of trees, minimum node size (MNS), and number of cuts.



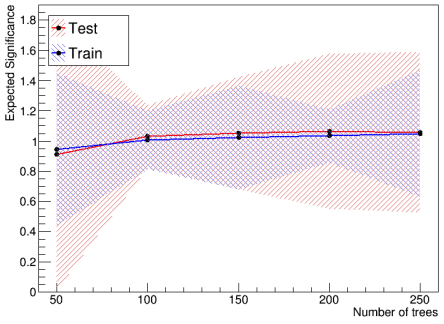
(a) Minimum node size



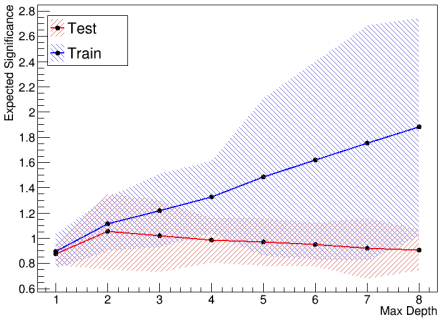
(b) Minimum node size



(c) Number of cuts



(d) Number of trees



(e) Max tree depth

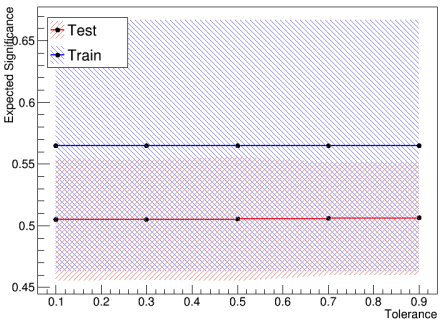
Figure 1: Effects of different BDT hyper-parameters on the significance.

The algorithm was found to be the least sensitive to the minimum node size parameter. In Figures 1a and 1b, you can see that there is no significant change in efficiency as the MNS changes. The algorithm however becomes less reliable at certain values as shown by larger errors for the test data below 6 and above 9. Number of cuts was another parameter that didn't affect the calculated significance. In Figure 1c you can see no change to the significance value as the number of cuts increases. The only values that should be avoided for this parameter are at the low end. In the graph, it is clearly visible that the errors increase below 15 cuts, making the algorithm unreliable in that region. A very similar trend can be seen in Figure 1d, which shows the effect of the number of trees on the calculated significance value. This time however, in addition to increased errors at low values of the investigated parameter, you can also see the significance value drop by a small amount. Based on this, the number of trees should be kept anywhere above 100.

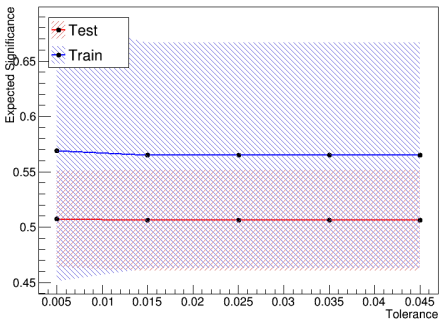
The last parameter is the maximum tree depth. Figure 1e shows the effects of this parameter on the LLR value. Looking at the Training data first, the efficiency seems to increase with the parameter, but at the same time so do the errors which makes large values less reliable. Simultaneously, the test data doesn't show the same improvement as the parameter is increased, and instead it peaks at 2 and slowly decreases at larger values. The mismatch of significance values for the test and training data at larger values of the maximum tree depth is an example of overtraining. Overtraining happens when the algorithm becomes too specialised in a specific set of data, and as a result underperforms when new data is presented that is a lot different. Due to this correlation between overtraining and maximum tree depth, values above 3 should not be used. Additionally, the significance is significantly lower at the lowest value of 1, and so the optimal values for this parameter are 2 and 3.

For Support Vector Machines, the investigated hyper parameters were Gamma, Tolerance, and Cost. Figures 2a and 2b shows the effect of tolerance on the efficiency of the algorithm. There is no visible effect across the tested regions that would suggest any sensitivity to this parameter. It can be assumed that tolerance doesnt affect the efficiency at all. The next parameter to be investigated was Gamma. In Figure 2c, it can be seen that the efficiency drastically drops as the value approaches zero. The peak significance seems to be around 0.5 gamma, and above that value the significance starts slowly dropping.

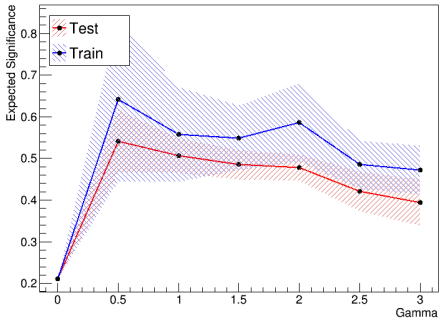
The Cost parameter was investigated over a large range of values. Firstly, the effects of large values of Cost on the efficiency was tested, and the results can be seen in the first graph in Figure 2d. There are no large changes in



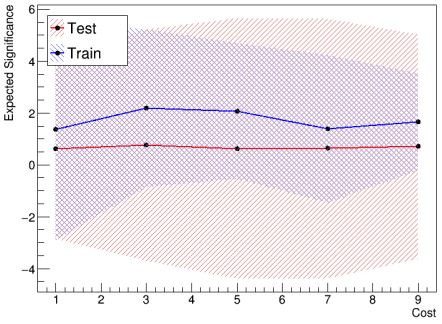
(a) Tolerance



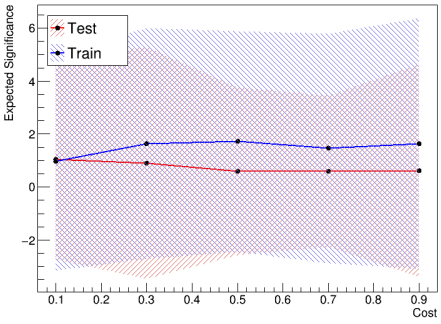
(b) Tolerance



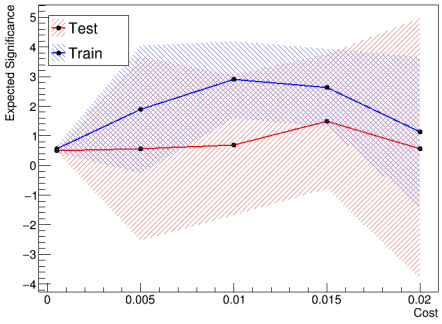
(c) Gamma



(d) Cost

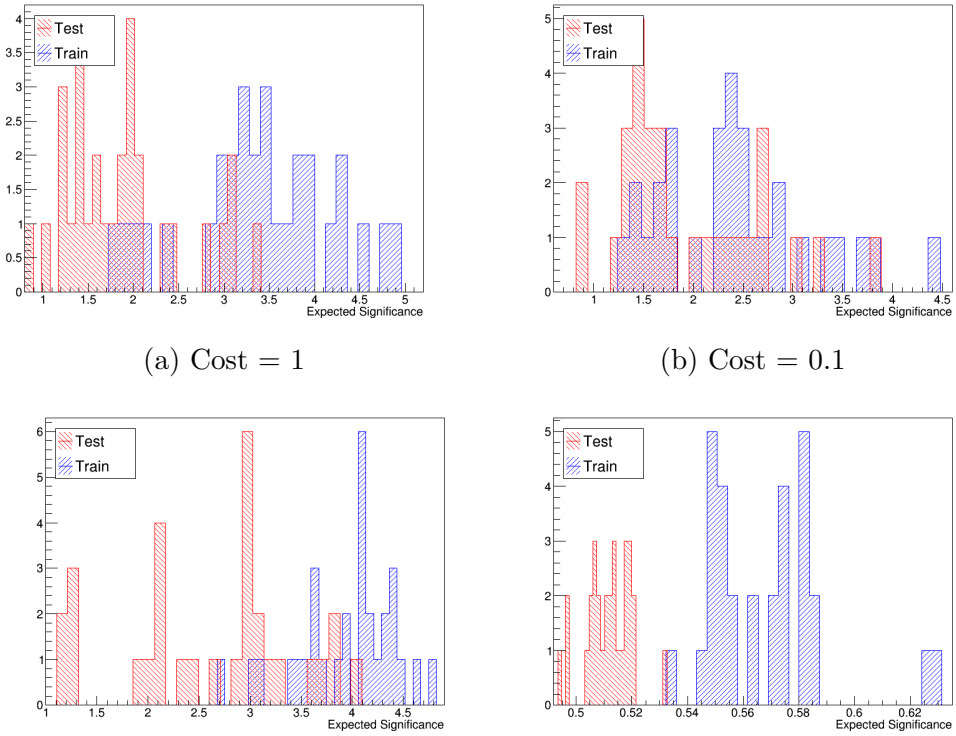


(e) Cost



(f) Cost

Figure 2: Effects of different SVM hyper-parameters on the significance.



(a) Cost = 1

(b) Cost = 0.1

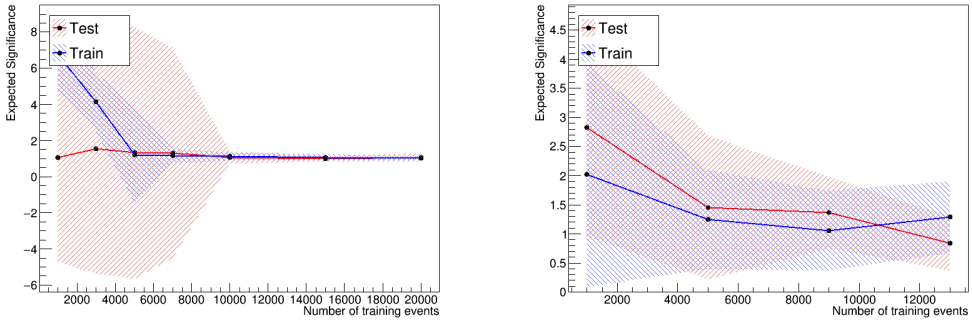
(c) Cost = 0.02

(d) Cost = 0.0005

Figure 3: Histograms showing significance values across all Jackknife iterations for different Cost values.

significance across the entire range shown in this graph, so it can be assumed that it is not sensitive to the changes in cost at high values. The second graph in Figure 2e shows a similar trend, with the exception of the test and train lines meeting at the value 0.1 of Cost. This is looked at further later in the discussion. The graphs suggest that the algorithm is not overtrained, as the errors mostly overlap. Since the only noticeable change so far has been at the smallest value investigated of Cost, a further investigation was carried out into values even smaller. In Figure 2f, there is some visible variation in significance, especially in the training data, however there is no clear improvement in performance. Additionally, errors disappear when the Cost value approaches zero, however the significance also drops.

In Figure 3, histograms of individual efficiency values from each jackknife sample are shown. Looking at Figure 3a and 3c, it is clear from the differences in significance that despite what graphs from Figure 2 show, the algorithm was overtrained after all. Looking at histograms of all investigated values

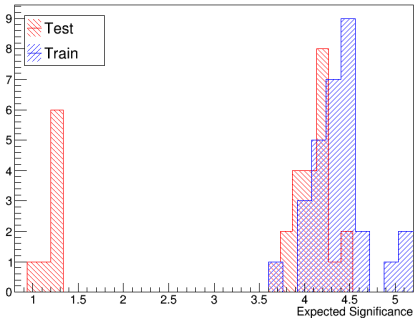


(a) BDT (b) SVM

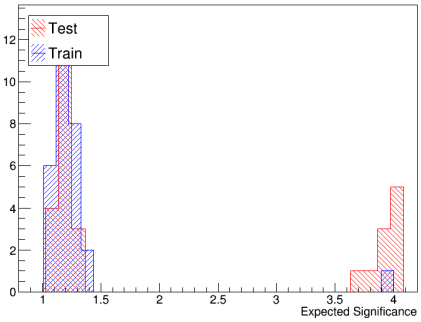
Figure 4: Graphs showing the effects of the number of training events on the Significance of an algorithm.

of cost, the best value to avoid overtraining is 0.1 as shown in Figure 3b. At that value of cost, test and train data have similar variance and peaks. As the value of cost moves away from 0.1 however, the algorithm becomes increasingly overtrained. Additionally, 0.1 is the value of cost where the test and train lines met in Figure 2e. Lastly, in Figure 3d the behaviour of the algorithm is shown as cost approaches 0. This is where the test and train lines seem to meet in Figure 2f, and the errors disappear. Although the train and test data doesn't overlap here, there is a very small difference between them which means that the algorithm isn't overtrained at this low value of cost. However it has a much worse performance than any other value investigated, which is why is should not be used over the previously mentioned cost value of 0.1.

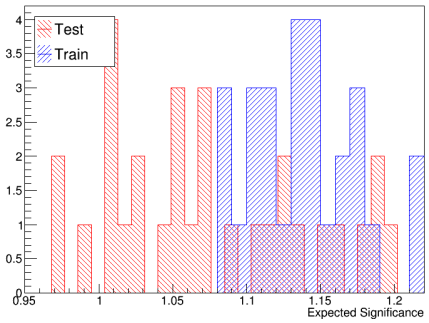
The two algorithms are compared based on their performance on different amounts of training data used. Figure 4a and 4b show the performances of BDTs and SVMs respectively. In the graph showing BDT performances, the efficiency doesnt change with the number of training events. However, the errors increase drastically below 10000 events, making the algorithm unreliable with a small amount of training data. Additionally, below 5000 events the algorithm is prone to overtraining which is clear from the separation between the train and test lines. It is important to note that the errors in this graph are not representative of the actual behaviour of the algorithm. Figure 5a shows the histogram of all calculated efficiencies for 3000 training events. In that histogram, you can see that the spread is not uniform, and instead you can see 2 peaks in the test data, where one overlaps with the peak for the training data. As the number of training events increases, the efficiency



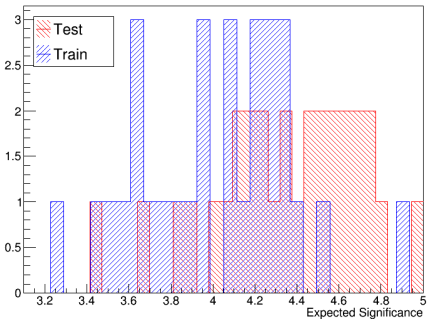
(a) BDT efficiency at 3000 training events.



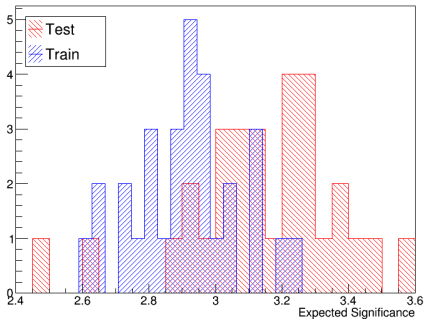
(b) BDT efficiency at 5000 training events.



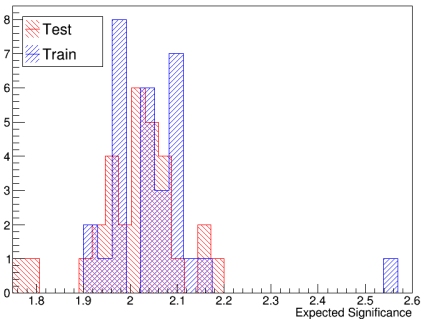
(c) BDT efficiency at 10000 trainig events.



(d) SVM efficiency at 1000 trainig events.



(e) SVM efficiency at 5000 training events.



(f) SVM efficiency at 13000 training events.

Figure 5: Histograms showing the effects of the number of training events on the Significance of an algorithm.

of the training data lowers to match the test data as shown in Figure 5b, which displays the histogram for 5000 training events. Once at least 10000 events are used, both the train and test data begin to show correct behaviour where there is only one peak, as seen in Figure 5c . Another thing to note is that in Figure 4a, the errors go below 0 when a small amount of training data is used. However, it is not possible to achieve a negative significance, and the histograms show all data is above zero. This is a situation where the Jackknife Variance Estimate fails, as it assumes that the spread of data is symmetrical, but that is not always true as shown in Figure 5b.

SVMs on the other hand turned out a lot more well behaved, which can be seen by looking at Figures 5d, 5e, and 5f. The histograms show more standard spread of data with a single peak. Both the test and train data show this behaviour and the peaks are very close to each other, with most of the data overlapping. This shows that no overtraining is present. The average efficiency is higher at lower amounts of training data, however the errors are also larger in that area. Lastly, in Figure 5f one point of training data can be seen away from the rest with a much higher significance of 2.55. This is the only behaviour displayed by SVMs that may be considered strange, however this could just be a statistical fluctuation.

Comparing the two algorithms together when a low number of training events is used, SVMs outperform BDTs. This is due to BDTs pathological behaviour in regions below 10000 events, making it completely unreliable. As a result, SVMs are a better choice there despite the large errors, though still not ideal. At 10000 events and above, BDTs have a consistent performance that SVMs dont, with their much smaller errors.

7 Conclusion

The aim of this investigation was to find which of the two machine learning algorithms would perform better in a particle physics environment. High energy physics produces large amounts of data on a regular basis, and so it is important to have efficient ways of analysing data. Machine learning methods are already used in HEP environments such as CERN, however it is important to investigate how reliable these algorithms are, and what are the optimal ways to tune them, as a badly tuned algorithm can give varied results with each training. Here, the results have shown that with a higher amount of training data, the BDTs are a lot more reliable. Additionally, its parameters appeared much easier to optimise than that of SVMs, which in the end still showed large errors in the results. SVMs can however perform better when little training data is available, as BDTs have proven to be

unusable in those situations. When it came to the relative performance, both algorithms performed similarly.

Due to the nature of particle physics research, the amount of data available for training would always be large, and so a situation where an SVM would need to be used over a BDT would never arise. Therefore, unless the hyper parameters for SVMs can be tuned to give better efficiency values and smaller errors, BDTs are the better option. If more time had been available for this investigation, SVMs could have been fine tuned to perform better and potentially give a different conclusion.

Furthermore, a different method could be used to compare the two algorithms. In the results and discussion section, there were numerous occasions where the graph of a parameter versus the efficiency would not be representative of the actual behaviours of the algorithm. This is because the value of the standard error doesnt show any information about the original shape of the distribution. As a result, details such as multiple peaks, or skewed distributions end up going unnoticed. If the graph had been plotted as a scatter graph, instead of calculating the errors, those details would be more easily visible in the final results.

Further research could be performed on those algorithms to ensure that optimal hyper parameter values have been found. If no improvement can be made to the performance of BDTs and SVMs, the next algorithm to investigate would be Neural Networks which is popular in HEP research. Additionally, different methods instead of the Jackknife could be used to carry out the investigation. One suggestion would be the Bootstrap which is a technique similar to the Jackknife, but more powerful and often considered more appropriate for physics research than the Jackknife.

8 References

[1] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, H. Voss, 2013, Toolkit for Multivariate Data Analysis with ROOT, p. 104-116
[2] Y. Caodou, 2013, Boosted Decision Trees and Applications
[3] A. Vossen, 2008, Support Vector Machines in High Energy Physics
[4] M.. Sahin1, D. Krcker, I.-A. Melzer-Pellmann, 2016, Performance and optimisation of support vector machines in high-energy physics classification problems
[5] A. Jeager, 2015, Computation of Two and Three Dimensional Confidence Regions with the Likelihood Ratio
[6] G. Ranucci, 2012, The profile likelihood ratio and the look elsewhere effect in high energy physics
[7] A.I. McIntosh, 2016, The Jackknife Estimation Method
[8] J.E.Gentle, 2002, Elements of computational statistics, p. 74-76