UNIVERSITY OF CAPE TOWN
DEPARTMENT OF COMPUTER SCIENCE

Computer Science Honours
Final Paper
2015

Title: A Comparative Analysis of Boosting and Artificial Neural Networks for Particle Identification at ALICE.

Author: Jed Boyle

Project Abbreviation: ALIRAD

Supervisor: Patrick Marais

Category	Min	Max	Chosen
Requirement Analysis and Design	0	20	0
Theoretical Analysis	0	25	0
Experiment Design and Execution	0	20	20
System Development and Implementation	0	15	10
Results, Findings and Conclusion	10	20	20
Aim Formulation and Background Work	10	15	10
Quality of Paper Writing and Presentation	10		10
Adherence to Project Proposal and Quality of Deliverables	10		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	0	10	0
Total marks		**80**	**80**

A Comparative Analysis of Boosting and Artificial Neural Networks for Particle Identification at ALICE.

Jed Boyle
Computer Science Department
University of Cape Town
BYLJED001@myuct.ac.za

ABSTRACT

At ALICE (A Large Ion Collider Experiment) the transition radiation detector is used to identify subatomic particles. In 2017 ALICE will be upgraded and the current particle identification, based on artificial neural networks, will no longer suffice. This paper investigates the applicability of boosted machine learning models rather than artificial neural networks for ALICE's upgraded system. This is accomplished by training a selection of boosting and artificial neural network models on simulated data and comparing the particle identification accuracy. The tests indicate that using AdaBoost to boost artificial neural networks yields the best performing boosting algorithm but that this performs worse than an artificial neural network using a single hidden layer and the sigmoid activation function. Consequently, the paper concludes that an artificial neural network should be used for particle identification at ALICE after the upgrade.

1. INTRODUCTION

A Large Ion Collider Experiment (ALICE) is the dedicated heavy ion experiment at the Large Hadron Collider (LHC). ALICE collides heavy ions - the nuclei of heavy particles - at nearly the speed of light [10]. In this paper we consider how to identify the particles that are emitted from these collisions, rather like subatomic shrapnel. ALICE has been operating since 2010 and it is scheduled for an upgrade in 2017 [8]. Most importantly, the upgrade will increase the rate and momentum of particle collisions improving the quality of results for current experimentation and allowing for new experimentation [8]. The new system will also impose new hardware requirements. As a result the current particle identification framework needs to be redeveloped.

Currently, artificial neural networks (ANN) are used for particle identification at ALICE [11]. A similar experiment, MiniBooNE at Fermilabs, uses boosting - a technique of combining many weak classifiers into a single strong classifier - to identify particles [12]. After the upgrade the new hardware at ALICE will require that the information used for particle identification gets compressed into six 8-bit values, which is currently not necessary at either ALICE or MiniBooNE. Consequently, the systems now in use at ALICE and MiniBooNE will not be compatible. This paper compares the feasibility of boosting as a basis for the new particle identification system at ALICE as compared to ANNs.

The construction of these 8-bit values, know as tracklet IDs, is studied in a companion paper.

To compare boosting and ANNs, a variety of adaptive boosting and feed-forward ANN algorithms are implemented. Adaptive boosting and feed-forward ANNs are the base on which the boosting and ANN class of algorithms is built. This means that the algorithms implemented in this project will give an indication of how well each class of algorithms would perform at ALICE after the upgrade. The algorithms are tested on simulated data generated by AliRoot - ALICE's software package.

This paper concludes - based on a wide range of experiments using both boosting and ANNs with different parameters and datasets - that ANNs will be more adept than boosting at identifying particles at ALICE after the upgrade.

The structure of the paper is as follows. Section 2 explains the background information relevant to ALICE and the algorithms that were implemented. Next, in section 3, we examine other research that relates to this project. Following this, in section 4 the implementation details and framework are explained. Lastly, section 5 reports and discusses the results of the experiments.

2. BACKGROUND

This section provides an introduction to the ALICE experiment and the algorithms used in this project. There is an emphasis on the specification of the boosting algorithms because they were implemented from scratch for this paper.

2.1 ALICE

The LHC is used to study conditions similar to those at the very beginning of the universe. The ALICE experiment at the LHC collides heavy ions to create quark-gluon plasma - an extremely hot emulsion that existed for a millionth of a second after the universe's creation before cooling to form the fundamental building blocks of matter [10]. Each collision emits thousands of particles such as electrons, protons, pions, muons and kaons [11]. Due to the number of particles and the complexity of the classification task, machine learning algorithms are needed.

Experiments at ALICE that require particle identification have very high noise to signal ratios [8]. As a result many

© 2015 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

collisions - in some cases more than the current system can produce - are needed to obtain statistically significant data [8]. This problem is going to be solved in 2017 when ALICE is scheduled for an upgrade.

The information used for particle identification at ALICE is recorded by the transition radiation detector (TRD) [11]. The TRD is a cylindrical detector that surrounds the point of collisions and records the temporal charge emitted from particles passing through it. The TRD cylinder is made up of segments called stacks that consist of six layers each. Every layer has a gas chamber which is used to record the amount of charge emitted by particles passing through it. Each layer is divided into 144 vertical columns and 12 or 16 horizontal rows [1]. A particle may traverse all six layers of the TRD, depending on its angle of refraction. When a particle traverses a layer of the TRD the amount of charge it emits is recorded across four of the vertical columns and a single row for 27 time bins. These charge values are known as ADC values. The components of the TRD are shown in figure 1. From the perspective of the particle identification algorithm, the result of a particle traversing the TRD is six matrices of ADC values, each containing the amount of charge deposited by the particle across 27 time intervals in four columns of the TRD layer. Each of the matrices is known as a tracklet and combine to form the information used to identify a particle track through the TRD.

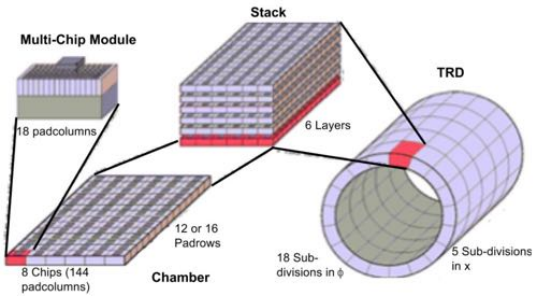

Figure 1: This figure shows the breakdown of the TRD into its components.

The current particle identification system at ALICE can use all the data contained in a tracklet[11]. However, after the upgrade in 2017 the information contained in the tracklet needs to be compressed into a single 32-bit word of which 8-bits, known as the tracklet ID, will be used for particle identification. This means that after the upgrade the particle identification system will have to use the 8-bit tracklet IDs from each of the six layers of the TRD to identify particles. As a result, the current system will no longer work.

The best technique for converting the tracklet information into a single 8-bit tracklet ID has not yet been established. This computation of a robust tracklet ID is the focus of a companion paper which looks into using a weighted summation of the ADC values.

This paper seeks to develop an accurate method to distinguish pions, short-lived particles made up of quarks and anti-quarks, from electrons, negatively charged elementary particles, using their deposited charge. In figure 2 the average deposited charge, ADC value, of electrons and pions is shown for each time bin. At ALICE more than just these two particles will need to be differentiated but at high momenta pions are similar to other particles so they are good surrogates for testing [11].

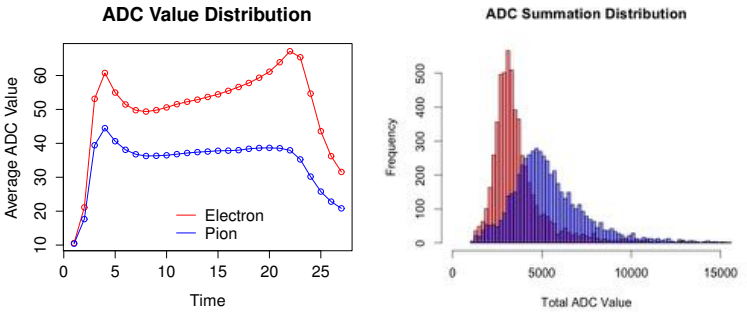

Figure 2: The plot on the left shows the average ADC value of pions and electrons for each time bin. The histogram on the right shows the frequency of the total ADC values for tracklets of each particle type.

AliRoot is the ALICE computing framework written in c++. It is used for data simulation, event reconstruction, detector calibration, detector alignment, visualization, and data analysis [11]. The data used in the project was simulated using AliRoot.

2.2 Decision Trees

A classification decision tree is a supervised learning technique modelled as a binary tree where the leaves are labelled with a class and the internal nodes represent a split of the predictor space [4]. Each split of the tree defines a partitioning of the training data. Picking a split is done using the Gini Index, which is a measure inversely proportionate to the percentage of a partition that belongs to a single class[4]. The higher this percentage the better and therefore the lower the Gini Index the better the split of the training data.

The tree is constructed from the root and recursively picks a partition of the predictor space which results in the largest decrease in the Gini Index until a stopping criteria is met[4].

Each internal node of the tree represents a split of the data, with respect to a predictor. As a result calculating the mean Gini Index decrease that each predictor creates across all of the nodes of the tree that it occurs in can be used to determine the importance of the predictor.

2.3 ANNs

ANNs are a popular supervised learning technique loosely based on the principles of a brain [6].

An ANN is a weighted directed graph made up of computing nodes called perceptrons. Each perceptron outputs on its outgoing edges the result of an activation function evaluated on the weighted sum of incoming edges[7].

In this project feed forward ANNs are used. The topology of the perceptrons in these networks has the perceptrons organised in successive layers[6]. Each layer is fully connected to the previous layer and there are no lateral connections between layers[6].

The learning process determines the weights of the edges. The partial derivative of the error function with respect to an edge weight is calculated and used to alter the weight of the edge in such a way that the error is reduced. This is

known as gradient descent. The error is propagated backwards through the layers of the network. This is know as back propagation [6].

The activation function considered in this project are the following.
Sigmoid:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Fast-sigmoid:

$$f(x) = \frac{1}{1 + |x|}$$

Tanh:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

Rectifier:

$$f(x) = \max(0, x)$$

2.4 Boosting

Boosting falls into a family of machine learning algorithms called ensemble algorithms. Ensemble algorithms construct a strong learner from a set of weak learners - which only need to be slightly more accurate than random [3].

Boosting algorithms maintain a weighting of the training data that is used to specialise weak learners [3]. The weight of more difficult training examples is increased, encouraging more weak learners to focus on them. The ease with which other weak learners identify a training instance is used to measure its difficulty [3].

Suppose we have T weak learners h_t then the strong learner h, performing binary classification, is defined as [3]:

$$h(x) = \text{sign}\left\{\sum_{t=0}^{t \leq T} \alpha_t h_t(x)\right\}$$

Each h_t is trained with respect to a weighting of the training data $w_{t,i}$ on training instance x_i. The weak learner weighting α_t determines the impact that h_t has on the overall classification.

A boosting algorithm is defined by how it calculates α_t and $w_{t,i}$. AdaBoost was the first implementation of boosting that solved the practical problems of early attempts [3]. It has become the template for the entire class of algorithms and is still a very effective classifier [9] [12] [13]. In this project we consider AdaBoost, ϵ-Boost and ϵ-LogitBoost.

The general structure of boosting algorithms is as follows. Set $w_{t,i}$ to $\frac{1}{n}$, where n is the number of training instances, and train h_1. Use h_1 to determine which training instances are hard to classify and increase their weights. Use the new data weighting to train h_2 and repeat this process for the remaining T weak learners [3]. In the following subsections the specifics of each of the boosting algorithms implemented for this project are explained.

2.4.1 AdaBoost

In AdaBoost the weights are updated during the training process by the following rule [3]:

$$w_{t+1,i} = \frac{w_{t,i} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}}{Z}$$

Z is a normalisation factor used to ensure that $\sum w_{t,i}$ remains equal to one.

We define α_t, the weak learner weighting of h_t, as [3]:

$$\alpha_t = \frac{1}{2} \ln(\frac{1 - \epsilon_t}{\epsilon_t})$$

where ϵ_t is the error of h_t and is defined as [3]:

$$\epsilon_t = \sum_{h_t(x_i) \neq y_i}^{i \leq n} w_{t,i}$$

Naturally, α_t and ϵ_t are inversely proportional. This makes α_t a useful measure of how valid and novel the classification of h_t is.

2.4.2 ϵ-Boost

ϵ-Boost requires picking a constant ϵ value which is in the order of 0.01. For each weak learner $\alpha_t = \epsilon$. The training weights are updated as follows [12]:

$$w_{t+1,i} = w_{t,i} \times \begin{cases} 1 & \text{if } h_t(x_i) = y_i \\ e^{2.\epsilon} & \text{if } h_t(x_i) \neq y_i \end{cases}$$

2.4.3 ϵ-LogitBoost

ϵ-LogitBoost also requires picking a constant ϵ value, which is in the order of 0.01, and $\alpha_t = \epsilon$ for each weak learner. Define, a partial classification h'_t as [12]:

$$h'_t(x) = \sum_{i=0}^{i \leq t} \alpha_i h_i(x)$$

The partial classification is used to define the weight update rule as follows [12]:

$$w_{t+1,i} = \frac{e^{-y_i h'_t(x)}}{1 + e^{-y_i h'_t(x)}}$$

where y_i is the label of training instance i.

2.4.4 Evolutionary Weighting

The boosting algorithms' weak learner weighting can be found using an evolutionary search [5]. Any of the previously mentioned algorithms' training data weighting scheme can be used in conjunction with evolutionary weighting of weak learners.

There are many variations of evolutionary algorithms. The key concepts are a fitness function, selection operators and variation operators which drive a population of possible solutions through multiple generations towards an optimal solution [2]. Variation operators combine and mutate individuals of the population creating novelty and diversity [2]. The selection operators choose individuals using the fitness function that are then used by the variation operators. The variation operators construct a new population from these individuals. If the selection and variation operators are well chosen each population generation will have an increased fitness [2]. This process is repeated until a stopping criteria is met.

2.5 k-Fold Cross Validation

k-fold cross validation is a technique used to determine how well a model will perform on unseen data. To achieve this the data is divided into k segments and the model is trained on $k - 1$ segments. The remaining segment is used to test the model. This is done k times using different $k - 1$ segments each time. Then the model's performance on the unseen data segments is averaged over the k segments.

3. RELATED WORK

This section gives the reader a brief overview of other studies that relate to this project.

Many of the classical particle identification techniques, such as truncated mean and cluster counting, have been used at ALICE [11]. Before implementing ANNs, likelihood based methods were most successful. These techniques use Bayesian statistics to determine the likelihood of a detector reading being produced by a certain particle to make classifications [11].

As explained in the background section, this project aims to identify particles by their deposited charge. Of the ANNs currently being used at ALICE, the one that uses deposited charge has eight input nodes, two hidden layers and an output node for each type of particle being identified [11]. There are 15 nodes in the first and five nodes in the second hidden layer. Each tracklet is identified by the ANN and likelihood based methods are used to combine the results into a single classification. The tracklet's data is divided into eight segments and the total deposited charge of each segment is used as an input for the ANN. It is shown that increasing the number of segments improves the network's accuracy [11]. This is not surprising but it is concerning because the construction of the tracklet ID in this project is the equivalent of using a single segment. Using an ANN improved the original likelihood based methods by three to four times [11].

The current ANN is simple and the author makes no guarantees that it has the optimal configuration [11]. It seems that it could quite easily be improved. The main issue with the current particle identification implementation, which is what this project deals with, is the fact that it isn't compatible with the new hardware specifications - requiring particle identification be done on 8-bit tracklet IDs.

Another gap in the literature is that there is no analysis of how well boosting performs at ALICE. A detailed analysis of the performance of boosting algorithms for particle identification at MiniBooNE has been studied [12] [13]. At MiniBooNE ϵ-LogitBoost performed worse than Adaboost and ϵ-Boost [12]. The major difference in performance between AdaBoost and ϵ-Boost was AdaBoost's ability to better utilise fewer weak learners [12].

The comparative performance of boosting and ANNs at MiniBooNE has also been studied [9]. At MiniBooNE boosted decision trees had better performance than ANNs on all tests and in some cases it improved classification accuracy by 80 percent [9]. Boosting also proved to be less sensitive to noise in the data [13].

The improved accuracy was in part due to the fact that boosted decision trees are better at utilising many explanatory variables[9]. At ALICE, after the upgrades, there will be six or less explanatory variables so this gain may be lost. Also the particles at MiniBooNE are not detected using a transition radiation detector. Given these facts it is hard to extrapolate the results of the MiniBooNE studies to ALICE's upgraded system .

One issue with AdaBoost is that its greedy weak learner weighting algorithm outputs an inefficient weighting [5]. To solve this problem an evolutionary search of the weak learner weight space can be used [5]. This technique improves the accuracy of AdaBoost and can be used to prune learners with low weighting - an indication that their prediction power isn't very strong[5]. Pruning these weak learners can improve classification speed and accuracy [5].

Unfortunately, evolutionary weighting has only been tested on AdaBoost [5]. This means we are unable to determine what the effect of evolutionary weighting is going to be on ϵ-Boost and ϵ-LogitBoost. Secondly, the experimentation is done on a facial recognition problem rather than a particle identification problem [5].

From the related work it is apparent that ANNs and boosting have been successfully used for particle identification experiments. However, these experiments are different from the type of problem ALICE's upgrade will present. Therefore, this project aims to determine how well they will perform at ALICE with the new hardware requirements.

4. FRAMEWORK

The key goal of this project's software was to provide a robust and flexible testing framework that could be used for experimentation in this project and others. The software also had to be compatible with AliRoot should it at some point become necessary to integrate the two frameworks. The framework was developed using c++ and the Shark machine learning library.

To make the software an effective testing framework it was developed so that parameters could easily be changed and results easily recorded. The parameters have clear names, are easily set using function arguments and functions are stable using all combinations of parameters. To get performance information, models can be passed to the CrossValidation class, which uses k-fold cross validation to determine how well the model will perform on unseen data.

The hierarchy of the machine learning classes in the framework is drawn in figure 3. Apart from these classes the framework includes a data processing python script, an optimiser class that contains the evolutionary functionality and a cross validation class for testing. The AbstractClassifier class defines the interface and important functionality, such as calculating error, that all models need. The boosting models are templated so that any model adhering to the AbstractClassifier interface can be boosted. This is convenient because the same boosting implementations can be used for many different weak learners. If in the future other models, such as support vector machines, are added they can also be boosted if they adhere to the AbstractClassifier interface. It also makes it convenient to use the framework because all the models adhere to a common interface. The current system includes the NetworkClassifier and TreeClassifier class which can be boosted.

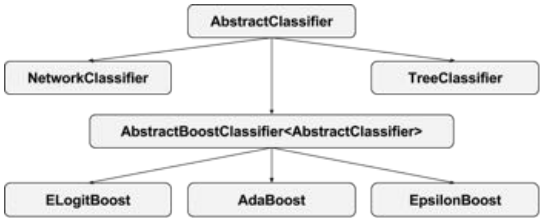

Figure 3: An overview of the machine learning class hierarchy.

The machine learning framework uses data processing functionality and models from the open-source Shark machine learning library, which provides functionality for reading CSV data. The data generated from the AliRoot simula-

tions can be processed into CSV data using the Python data processing script developed with this project. Once Shark has read the data it is efficiently handled in batches and can be passed to any of the Shark models. In this project the Shark decision tree and feed-forward ANN are used. The project was implemented in c++ to make it compatible with AliRoot.

4.1 Implementation

4.1.1 Decision Trees

In the Shark library, decision trees are implemented in the CARTClassifier class. The CARTTrainer class is used to train these models. In this project the decision trees are going to be boosted and therefore the framework needs to be able to limit their size. The Shark decision tree trainer did not provide this functionality so the CARTTrainer used in this project has been altered so that the maximum depth of a tree can be limited. It does this by using the original shark implementation to grow the tree to full size and then removes nodes, in the reverse order they were added, until the tree is the correct size. The altered CARTTrainer and CARTClassifier classes are encapsulated in the TreeClassifier class.

4.1.2 Artificial Neural Networks

In the Shark library, feed forward ANNs are implemented in the FFNet class. Associated with this class is an optimiser class that trains the network. The NetworkClassifier class wraps both of these classes into a single class adhering to the AbstractClassifier interface. The interface was designed so that all the underlying functionality of the Shark ANN and optimiser could be accessed. To achieve this all the parameters are set in the constructor. This allows the class to adhere to the AbstractClassifier interface it inherits. Like the FFNet class the type of neurons that the network uses can be passed to the NetworkClassifier class as a template argument.

4.1.3 Boosting Algorithms

AdaBoost, ϵ-Boost and ϵ-LogitBoost were implemented from scratch in this project since Shark does not have any boosting features. The AbstractBoosting class implements the boosting algorithm. The AdaBoost, EpsilonBoost and ELogitBoost classes which inherit from the AbstractBoosting class contain the different weight updating functionality.

4.1.4 Evolutionary Weighting

Any boosting algorithm can be passed to the Optimiser class and it will use an evolutionary algorithm to re-weight the weak learners.

Each instance in the population is a vector of weights. The algorithm uses uniform cross-over meaning that two weight vectors are combined into a single offspring by randomly selecting a weight for each weak learner from one of the parents weights for that weak learner. Fitness proportionate parent selection is used. This means that two parents are sampled with respect to the individuals weights from the population. Ten percent of the previous population survives to the next generation and the rest is made up of the offspring with the best fitness. This is partial elitist survivor selection. The size of the population, number of generations, mutation probability and other parameters can be set using the class constructor.

4.2 Data

CERN has restriction on access to real data at the LHC so the data used in this project was generated using the simulation tools in AliRoot.

The AliRoot macros used to generate this data are part of the framework. The framework also includes a python script which processes the raw data into csv files that the machine learning classes can read.

The simulation macros generate data that isn't very clean. Some particles have erroneous detector values and not all particles traverse each of the six layers of the TRD. The python data script handles both of these cases. Particles with erroneous detector values are detected and removed. Particles that traverse fewer than six layers of the TRD have the missing values inserted using the mean values of the TRD layers that the particles did traverse.

We were unable to source the amount of data we would have liked. The data used in this project contains 2413 particles of which 1218 were electrons and 1195 were pions. Unfortunately, having a small amount of data limits the accuracy of the results in this paper and makes it difficult to compare this framework with the actual implementation being used at ALICE.

As mentioned previously, after ALICE's upgrade the data from each tracklet needs to be converted into a single 8-bit tracklet ID. The current ANN system sums eight segments of the ADC values and uses this as the input for the ANN [11]. So using the sum total of all ADC values truncated to 8-bits is a natural construction for the tracklet ID. This was the method used for generating tracklet IDs in this project. The distribution of the resulting tracklet IDs can be seen in figure 2. Converting the data into tracklet IDs was implemented in the python data processing script.

5. RESULTS

In this section the results of the experiments done in this project are discussed. All the algorithms were run on Amazon Web Service's compute optimised machines with four cores and 16GB of RAM. All of the experiments in this section were done using 10-fold cross validation, meaning that all of the results were recorded on unseen data.

5.1 Tracklet ID

The ability of the machine learning algorithms to classify each particle depends on the quality of the information communicated in the tracklet ID. A full analysis of this is done in the companion paper. This section aims to determine if the Gini Index provides a useful weighting of the ADC values. This was done with experiments that compare the accuracy of decision trees and ANNs on two different tracklet ID constructions - namely, a weighted summation using the mean Gini Index decrease as a weighting as well as an un-weighted summation.

Determining the mean Gini Index decrease was done using 200 trees each selecting a split from a random selection of five predictors. Preventing each tree from selecting a split from the same set of predictors means that a more varied set of predictors will be used for splits. This allows us to get a better measurement of the mean Gini Index decrease that each predictor results in [4].

In the left hand plot of figure 4 the mean Gini Index

decrease of the ADC data is plotted. The original two-dimensional ADC value matrix of each tracklet was linearised for these tests. Hence, in figure 4 the first 27 values represent the first column of the tracklet data and the next 27 values represent the second column and so on. From figure 4 we can tell that the two central columns are more important than the outer ones. Surprisingly, the column left of centre is in general more important than the column on the right of centre. It is also apparent from figure 4 that the latter time-bins have more descriptive power. This is not surprising given the shape of the average ADC values' for pions and electrons, plotted in figure 2.

The normalised mean Gini Index decrease for each ADC value was used as a weighting to construct a new data set of particles using a weighted summation for the tracklet IDs. This was done by normalising the mean Gini Index decrease and then truncating the summed total to 8-bits. To determine if the weighted tracklet IDs performed better we fitted various decision trees and ANNs to both data sets.

The results of fitting decision trees are in figure 4. Decision trees with maximum depth larger than five had a better training accuracy on the weighted data. The gain wasn't large but it does indicate that the concept of using a weighted summation to construct the tracklet IDs can improve performance.

The data using the mean Gini Index decrease to construct a weighted tracklet ID increased the training error of ANNs. This is not surprising given that decision trees were used to construct this weighting.

The results in this section prove that some of a tracklet's ADC values are more important than others. Additionally, using a weighted summation can improve the descriptive power of each tracklet ID. However, ANNs and decision trees need different weight distributions to improve performance. The mean Gini Index decrease weighting scheme works for decision trees but not ANNs.

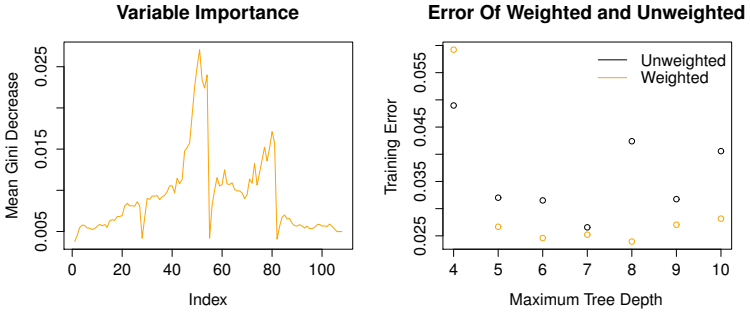

Figure 4: Plots showing the results of the mean Gini Index decrease weighting scheme.

5.2 Boosting

5.2.1 *Weak Learners*

Boosting algorithms are dependent on the performance of the weak learners being boosted. Selecting the learners is a balancing act. Boosting weak learners that are too complex over fits the data and boosting weak learners that are too simple results in a useless model. In this section of the paper we compare the performance of boosted decision trees and boosted ANNs. Boosted decision trees are a common construction and have been shown to be powerful classifiers [12] [9] [13]. ANNs have been shown to be accurate classifiers at ALICE [11]. The experiments in this section determine which weak learner, decision tree or ANN, and which configuration of weak learner will perform best at ALICE after the upgrade.

The performances of a decision tree and of a ANN are plotted in figure 5, which shows the CV error of the two models. The ANN has the same parameters as the ANNs that are boosted. The exact parameters are specified below.

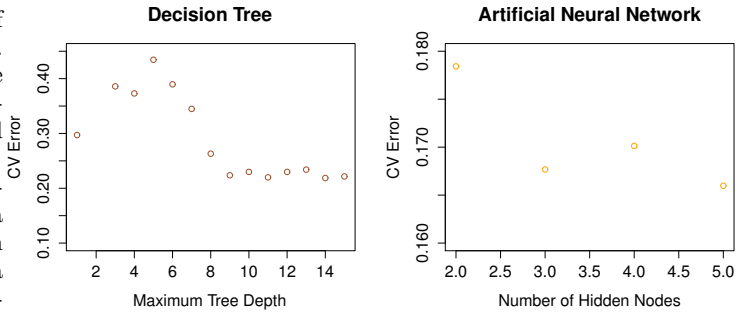

Figure 5: Plots comparing the performance of different sized decision trees and ANNs.

As we can see from figure 5 the optimal decision trees obtain a CV error in the region of 0.25. The optimal tree has a depth of nine. This means that restricting the tree's maximum depth to a number bigger than nine has no impact on the decision tree that is built. The plateauing of the decision tree's CV error seen in figure 5 illustrates this point.

The major difference between the individual ANNs and decision trees is the CV error. The CV error of the ANNs is much lower than that of the decision trees. Even simple networks with only two nodes and a single hidden layer perform better than the optimal decision tree.

Boosting algorithms, using ANNs and decision trees as weak learners, were implemented to determine which of the two is a better weak learner. All of the boosted models were trained using 200 weak learners. In boosting implementations the number of weak learners is usually in the region of 3000 [12] [13]. However, the training data generated for this project only contains 2413 particles. It would be unrealistic to have a similar number of weak learners and training instances. For this reason only 200 weak learners were boosted.

Experimentation was done with the maximum depth of decision trees varied between one and nine. As we can see from figure 5 increasing the maximum depth beyond nine would make no difference.

These experiments determined that deep decision trees are required at ALICE even though shallow decision trees, as can be seen from figure 5, are more accurate than random. Theoretically, this means they should work but this is not the case because boosting algorithms weight the training data. The initial learners in the boosting process are trained on a data set close to the original, as is the case in figure 5. As the algorithms change the data weights, more difficult training examples are selected resulting in more complex training sets for the weak learners. In some cases weak

learners are unable to infer valuable information from these complex samples, reducing accuracy. This is the case when the maximum tree depth is less than seven. An example when trees have a maximum depth of five can be seen in figure 6.

To determine if using more weak learners would solve this problem, models using 1000 shallow trees with a depth of less than five were trained but the same pattern of more learners increasing the CV error continued.

The boosted ANNs use the sigmoid activation function and 200 optimisation steps. Only a single hidden layer is used to limit complexity. Too-complex ANNs will over fit the data and increase training times. The size of the ANNs' hidden layer was varied between two and 15.

A summary of the results of boosted ANNs is plotted in figure 7. Unlike decision trees, all of the ANN sizes managed to improve performance.

The difference in shape of the curves of the boosted decision trees in figure 6 compared to the boosted ANNs in figure 7 is significant. It shows that boosted ANNs, in particular ones with simple hidden layers, learn more gradually and continue to improve over a larger range. By 50 weak learners the accuracy of boosted decision trees has plateaued but ANNs are still improving at the 200th weak learner. The fact that the ANNs learn more gradually indicates that they will generalise better [4]. Also the fact that they continue to improve accuracy for longer indicates that they will be better suited to boosting a large number of weak learners, which would happen the actual boosting implementation at ALICE.

Another advantage of ANNs is that simple network topologies with less than three hidden nodes can achieve similar performance to large trees with nine levels and more than a 1000 splits. This suggests that boosted ANNs will classify faster than boosted decision trees.

The experimentation done in this section indicate that simple ANNs with two or three hidden nodes and 200 optimisation steps will be the best performing weak learners at ALICE after the upgrade.

5.2.2 *Boosting Algorithms*

Boosting algorithms are defined by how they weight the training data and weak learners. Weighting specialises weak learners and balances their predictions. This section of the project aims to determine whether AdaBoost, ϵ-Boost or ϵ-LogitBoost would be the best performing boosting algorithm at ALICE. These algorithms were tested because they were effective at identifying particles at MiniBooNE [12] [13].

One of the advantages that boosting algorithms have is that there are almost no parameters that need to be chosen. For AdaBoost the user only needs to pick the number of weak learners. For ϵ-Boost and ϵ-LogitBoost the ϵ parameter needs to be chosen as well. Compare this to an ANN that can have very complex hidden layer topologies. In this project we used 200 weak learners for reasons mentioned in the previous section and ϵ was set to 0.01. This is a common choice for the ϵ value [12].

From the experimentation it became clear ϵ-Boost is equally or more accurate than ϵ-LogitBoost. For decision trees and big ANNs ϵ-LogitBoost has very similar performance to ϵ-Boost. However, ϵ-Boost performs better on small networks. This can be observed in figure 6 and 7.

Another disadvantage of ϵ-LogitBoost is that it takes longer

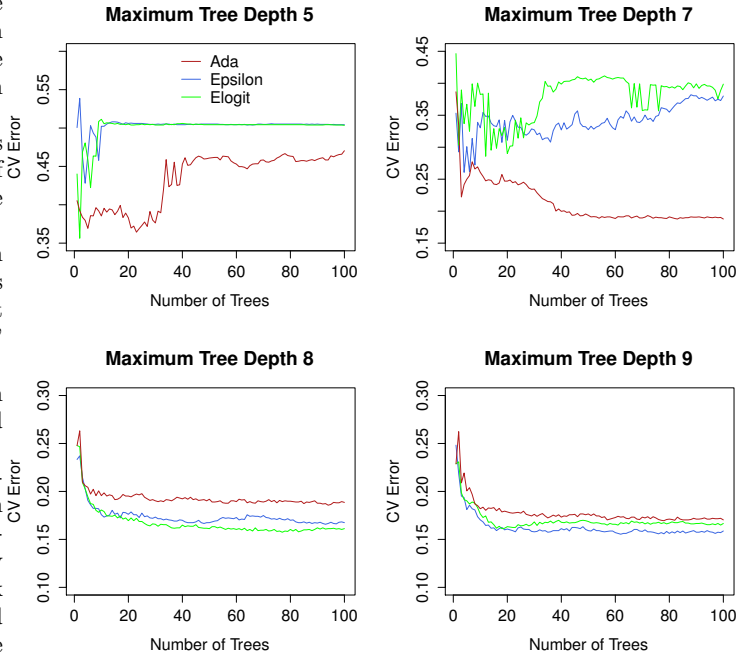

Figure 6: Plots showing the accuracy of different boosting algorithms using various tree sizes.

to train because of its training data weight updating rule. Each weak learner needs the partial classification of all the previous learners on each training instance to update the weights. AdaBoost and ϵ-Boost only need the classification of the single previous learner. For these reasons ϵ-Boost and AdaBoost would be a better choice than ϵ-LogitBoost for boosting at ALICE.

The next consideration is whether AdaBoost or ϵ-Boost is a better option. Both algorithms have very similar training and classification times so accuracy is the determining factor in choosing between them.

The key difference in accuracy of AdaBoost and ϵ-Boost is that AdaBoost is more accurate when boosting simple weak learners and ϵ-Boost is more accurate when boosting complex weak learners.

The better performance of AdaBoost on the simple weak learners is the result of its weak learner weighting algorithm. Unlike ϵ-Boost and ϵ-LogitBoost, which have a constant weak learner weighting, AdaBoost weights weak learners inversely to their error. This means that weak learners which struggle on complex samples and have large errors are weighted less, preventing them from dramatically impacting the classification. Simple weak learners are more likely to have large errors.

It is possible that AdaBoost's advantage when boosting simple weak learners comes from its weighting of the training data. It is hard to conclude that the data weighting has no impact but in the next section we look at evolutionary weak learner weighting and it becomes clear that the primary factor is the weak learner weighting. Figure 8 shows that weighting the weak learners in ϵ-Boost and ϵ-LogitBoost results in performance very similar to AdaBoost's. This indicates that the weak learner weighting is the key difference.

ϵ-Boost's improved accuracy compared to AdaBoost's on

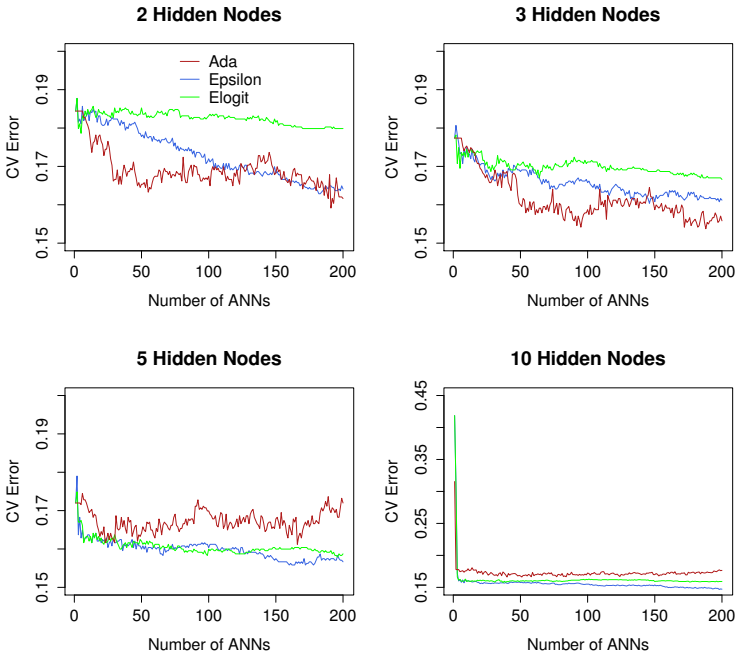

Figure 7: Plots showing the accuracy of different boosting algorithms using various network sizes.

complex weak learners is also the result of the algorithms' weak learner weighting. In AdaBoost the initial complex weak learners will have very low error rates and a large weighting resulting in them dominating the weak learner weighting and overall classification. In ϵ-Boost the weighting is fixed so the initial complex learners don't control the classification. This means that they can better utilise learners later in the boosting process.

These results are evident in figures 6 and 7. The performance difference resulting from AdaBoost's weighting of weak learners is most apparent in figure 6 when the maximum tree depth is seven. The figure shows that AdaBoost is increasing the performance on a weak learner that ϵ-Boost and ϵ-LogitBoost are unable to do.

In general, boosting algorithms with simple weak learners generalise better [4]. In this section it has been shown that AdaBoost is the best algorithm using simple weak learners because of its weak learner weighting. For this reason AdaBoost is likely to be the most effective algorithm for boosting at ALICE after the upgrade.

5.2.3 Evolutionary Weighting

In the previous section the importance of weighting weak learners became apparent. AdaBoost weights weak learners inversely to the weak learners' error and ϵ-Boost and ϵ-LogitBoost use a constant weighting for the weak learners. In this section we experiment with an evolutionary search for the weak learner weights. The aim is to determine if the weighting found by the evolutionary search improves accuracy and performance.

The experimentation was done using the evolutionary algorithm described in the Framework section. The boosting algorithms were trained with 50 decision trees of various maximum depths. The evolutionary search was very slow so only 50 weak learners were tested. In the previous sections it became clear that most of the boosting algorithms' decrease of error happens within the first 50 learners so this should nevertheless be instructive.

The mutation probability was 0.1, the population size was 500 and the algorithm was given 200 generations to find an optimal solution. During experimentation it was determined that the population converged within 150 generations so the algorithm was given 200 generations to be safe. When testing various sizes of boosting algorithms it was discovered that having a population size 10 times the size of the number of weak learners balanced speed and accuracy. If the population was too small relative to the number of weak learners the population did not converge.

As we discussed in the previous section ϵ-Boost and ϵ-LogitBoost were less effective at boosting simple weak learners than AdaBoost because they do not weight weak learners. Therefore, using an evolutionary search of the weak learner weight space should improve ϵ-Boost and ϵ-LogitBoost when using simple weak learners. Indeed this was the case in our experiments. Figure 8 shows the increase in accuracy after evolutionary weighting on all three algorithms. The figure shows that ϵ-Boost and ϵ-LogitBoost using decision trees with maximum depth seven in particular benefit from evolutionary weighting.

Weighting the weak learners in ϵ-Boost and ϵ-LogitBoost improved performance and made them perform similarly to AdaBoost on simple weak learners. As result it seemed natural to test what would happen if ϵ-Boost and ϵ-LogitBoost were combined with AdaBoost's weak learner weighting algorithm. This was implemented and it did improve performance but not as much as the evolutionary weighting. This indicates that evolutionary weighting is more effective than AdaBoost's weak learner weighting algorithm.

On larger trees the increase in performance from evolutionary weighting is less. As can be seen in figure 8 evolutionary weighting decreases the performance of AdaBoost and makes almost no difference to ϵ-Boost and ϵ-LogitBoost when the maximum tree depth is nine.

According to the literature one of the advantages of evolutionary weighting is that the new weighting gives a better indication which of the weak learners are not providing novel classifications than the normal weighting algorithm does [5]. This means that these weak learners, with low weighting, can be removed. For smaller trees this was the case in this project's experiments as well. As we can see from figure 8 when the maximum tree depth is seven AdaBoost can have about 40 weak learners pruned before the CV error of the model drops below the original CV error. For ϵ-Boost and ϵ-LogitBoost pruning 49 weak learners still has a better performance than the previous unweighted boosted model. This is because, as can be seen in figure 6 for ϵ-Boost and ϵ-LogitBoost using trees with a maximum depth of seven without any weighting, increasing the number of weak learners reduces performance.

The experiments done in this section indicate that evolutionary weighting can be used to improve accuracy and determine which weak learners should be pruned. This was especially the case for ϵ-Boost and ϵ-LogitBoost using smaller trees.

5.3 Artificial Neural Networks

ANNs are currently being used at ALICE and have been

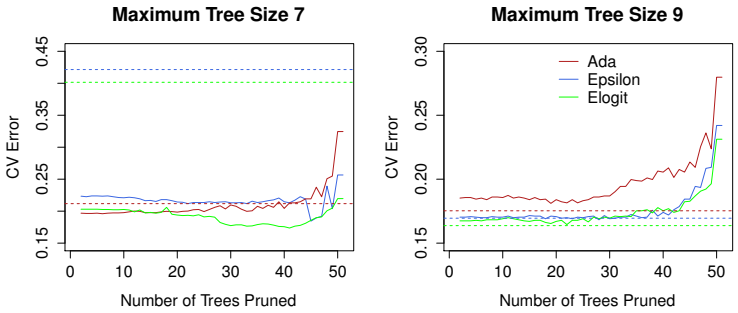

Figure 8: Plots showing the effects on the accuracy of different sized trees and boosting algorithms after evolutionary weighting of the weak learners. The dashed lines indicated the CV error of the respective models before evolutionary weighting.

shown to be strong classifiers [11]. Because of this ANNs are a candidate model for the new particle identification framework at ALICE. It is important to know the network topology and activation function that result in the best particle identification accuracy at ALICE.

For these experiments we used sigmoid, fast-sigmoid, tanh and rectifier activation functions. The networks were trained with two hidden layers that varied in size between one and 30. The current network has 17 nodes in the first layer and five in the second layer so this seemed liked a reasonable range. Each network was given 750 optimisation steps. Increasing this number began to over fit the models.

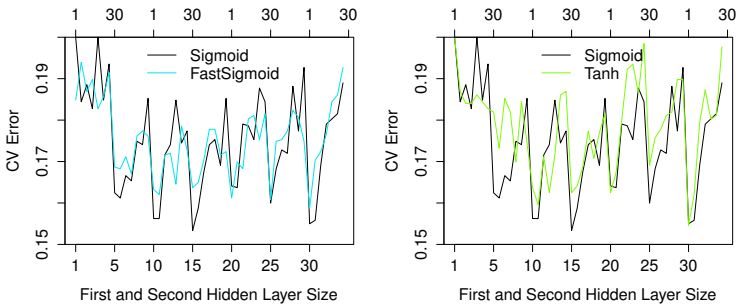

Figure 9: These plots show the CV error of ANNs with different activation functions and network topologies. The lower x-axis indicates the size of the first hidden layer and the top x-axis indicates the size of the second hidden layer.

The experiments showed that ANNs with small second hidden layers performed better. The plots in figure 9 show this. Each plot has an oscillating shape similar to a sin function. The valley of each oscillation occurs when the second hidden layer is small. The graph climbs until the second layer's size is reduced. This indicates that having a second layer is over fitting the data.

As we move along the x-axis the size of the first hidden layer increases and the valleys become deeper, especially in the case of the tanh and fast-sigmoid activation functions.

It became clear from the experiments that the sigmoid activation function was optimal. In figure 9 the performance of the rectifier activation was excluded because its performance was not nearly as good.

The best performing models have a CV error just above 0.15. All three activation functions graphed in figure 9 are able to achieve this level of accuracy. However, the sigmoid activation can do it with a less complex hidden layer. As one can see from figure 9 the CV error of the sigmoid activation function with 15 nodes in the first hidden layer is very similar to the other activation functions with 30 hidden nodes in the first hidden layer.

The general trend of the graphs also indicates that the sigmoid activation function is optimal.

The experiments comparing fast-sigmoid and sigmoid indicate that ANNs using fast-sigmoid are unable to learn the data as well as those using a sigmoid activation function. In figure 9 this is evident from fast-sigmoid's shallower valleys and lower peaks. The low peaks are a good thing because it indicates that the model doesn't over fit the data. However, a model that doesn't over fit is no good if it can't predict. This is the case with the fast sigmoid-activation function. We can tell this from the low valleys in figure 9.

As with fast-sigmoid, using the tanh activation function results in lower levels of accuracy than using the sigmoid activation function. This is evident in figure 9 by the shallower valleys of the tanh plot. However, unlike fast-sigmoid which doesn't over fit as badly as sigmoid on complex network topologies, tanh does. The high peaks of the tanh curve in figure 9 indicate this.

The experiments in this section have determined that a single hidden layer is best. The fast-sigmoid activation function is better at not over fitting the data than the other activation functions but is less accurate. Tanh over fits the data and is less accurate than using a sigmoid activation function. Therefore these experiments show that the sigmoid activation function performs best.

5.4 Comparing Boosting and Artificial Neural Networks

Various people have shown that boosting and ANNs can be used for particle identification [11][9][12][13]. This section of the paper will combine the observations obtained from the experimentation in the previous sections to determine if boosting or ANNs are more suitable for particle identification at ALICE after its upgrade.

In the previous sections we determined that using ANNs as weak learners performed better than decision trees did. We also found that boosting a simple ANN allowed the boosting algorithms to learn more slowly and prevented over fitting. Among the boosting algorithms AdaBoost performed best on simple weak learners as a result of its weak learner weighting scheme. Therefore, in this section AdaBoost with a thousand ANNs each having two nodes in the hidden layer was used to compare with an ANN.

In the ANN section we found that using the sigmoid activation function had better performance than a fast-sigmoid activation function and didn't over fit like a tanh activation function did. We also found that ANNs with complex second hidden layers over fitted the data. For this reason an ANN with a single hidden layer and sigmoid activation function was used to compare with the optimal boosting algorithm AdaBoost.

The performance of both these optimal models is shown in figure 10.

The most striking difference between the two models is the CV error. The individual ANN obtains error rates well

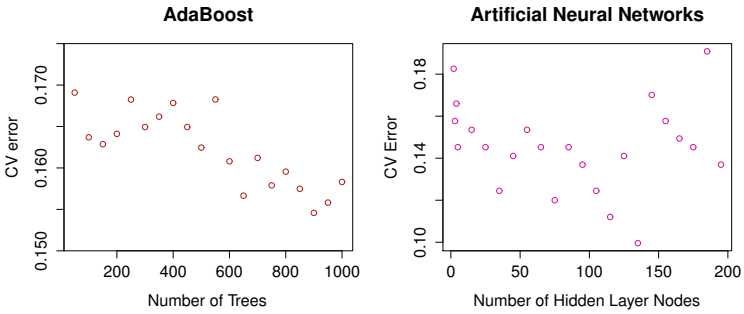

Figure 10: Plots comparing the optimal boosting and ANN algorithms.

below 0.15 which AdaBoost cannot do even with 1000 weak learners. There is a single ANN with 140 nodes in the hidden layer that gets less than 0.11 accuracy. The fact that this ANN does better than all the others with similar sized hidden layers indicates that this may be an outlier and not representative of how well it would perform on other data. However, even if we remove this point the general trend of the ANNs is achieving better CV errors than AdaBoost.

It appears that this type of ANN has achieved its optimal level of accuracy. It is fair to say this because using more than 150 nodes in the hidden layer is over fitting the data and increasing the CV error. Even with 1000 weak learners the trend of AdaBoost is down and it doesn't appear to be over fitting the data. So it could be that using more weak learners would eventually result in AdaBoost having better performance than the ANN. As mentioned in a previous section having 1000 weak learners and only 2413 particles is already unrealistic. Increasing the number of weak learners further would make this an even more unfair comparison. So it is safe to say that ANNs are more accurate for the type of problem posed by ALICE's upgrade.

A single ANN will also train and classify faster than AdaBoost. The single ANN was given 750 optimisation steps and AdaBoost used 200 optimisation steps per weak learner. This means that training the AdaBoost model required over 100,000 optimisation steps. These optimisation steps would have been faster but there are many more of them. AdaBoost also has to update weights. These facts results in the AdaBoost model training slower than the ANN. The speed of classification is also important. In the AdaBoost model each network had two hidden nodes meaning that classifying the entire model would require using 2000 perceptrons. The ANN has less than 200 perceptrons and will therefore classify faster. Even these rough analyses of the algorithms' running times make it clear that AdaBoost is computationally more costly.

The results in this section show that a single ANN is more accurate and computationally efficient than the optimal boosting algorithm AdaBoost for tackling the ALICE particle identification problem posed by the planned upgrade.

6. CONCLUSIONS

In 2017 ALICE is being upgraded [8]. The upgraded system will allow for new experiments and also provide better results for experiments with very high signal to noise ratios [8]. An important aspect of the experiments is identifying particles - in this paper electrons and pions - that traverse the TRD. The new system imposes hardware limitations requiring that each tracklet is identified by up to six 8-bit tracklet IDs. As a result the current implementation is going to be redeveloped.

In this paper we have tested various boosting models and ANNs on simulated data to determine which will be better suited to the upgraded system at ALICE.

From the set of boosting algorithms tested it was found that ϵ-LogitBoost was the worst performing. ϵ-Boost performed better than Adaboost on complex weak learners but worse than AdaBoost on simple weak learners. Since using simple weak learners generalises better AdaBoost was determined to be the best performing boosting algorithm.

Using ANNs with two or three nodes as weak learners performed better than decision trees. These small ANNs learnt slower and continued to improve performance for longer than the other weak learners tested.

As a result the best performing boosting algorithm was determined to be AdaBoost using ANNs with two nodes in the hidden layer.

ANNs using the sigmoid activation function learnt the data better than the fast-sigmoid activation function and over fitted the data less than the tanh activation function. ANNs with a single hidden layer performed best. As a result it was determined that ANNs with a single hidden layer using the sigmoid activation function was the best performing ANN.

The best performing ANN - single hidden layer and sigmoid activation function - was compared to the best performing boosting algorithm - AdaBoost using ANNs with two nodes in the hidden layer. The analysis shows that the ANN was more accurate and computationally more efficient in terms of training and classifying.

Considered collectively, our experiments indicate that after the upgrade at ALICE ANNs should be used for particle identification.

7. FUTURE WORK

This project concludes that ANNs are a better option than boosting for particle identification at ALICE. However, there are many variations of ANNs and a full investigation of these is an important next step before implementing a new particle identification framework at ALICE.

The accuracy of the entire model is also dependent on the tracklet ID. This paper has shown that it is possible to construct a more descriptive tracklet ID using a weighted sum. Further research should be done to construct more predictive tracklet IDs. Possibly evolutionary algorithms or ANNs could be used to calculate more sophisticated weightings.

8. ACKNOWLEDGEMENTS

The simulated data used for this project was sourced by Ryan Wong. The project was supervised by Associate Professor Patrick Marais from the University of Cape Town's computer science department. Thomas Dietel from the University of Cape Town's physics department co-supervised the project, providing assistance with the physics associated with it. The project would not have been possible without their valuable assistance.

9. REFERENCES

[1] T. Alice. Technical design report. *CERN/LHCC*, 21, 2001.

[2] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Springer Science & Business Media, 2003.

[3] Y. Freund, R. Schapire, and N. Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[4] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*. Springer, 2013.

[5] J.-S. Jang and J.-H. Kim. Evolutionary pruning for fast and robust face detection. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1293–1299. IEEE, 2006.

[6] S. Lek and J.-F. Guégan. Artificial neural networks as a tool in ecological modelling, an introduction. *Ecological modelling*, 120(2):65–73, 1999.

[7] R. P. Lippmann. An introduction to computing with neural nets. *ASSP Magazine, IEEE*, 4(2):4–22, 1987.

[8] T. Peitzmann. Alice upgrades. In *EPJ Web of Conferences*, volume 60, page 10003. EDP Sciences, 2013.

[9] B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor. Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 543(2):577–584, 2005.

[10] A. Trafton. Explained: Quark-gluon plasma. *MIT News*, 2010.

[11] A. Wilk. *Particle identification using artificial neural networks with the ALICE transition radiation detector*. PhD thesis, Münster (Westfalen), Univ., Diss., 2010, 2010.

[12] H.-J. Yang, B. P. Roe, and J. Zhu. Studies of boosted decision trees for miniboone particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 555(1):370–385, 2005.

[13] H.-J. Yang, B. P. Roe, and J. Zhu. Studies of stability and robustness for artificial neural networks and boosted decision trees. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 574(2):342–349, 2007.