An Analytical Comparison Among Various Multivariate Methods Used for Particle Identification

Yash Rana
Department of Physics, IIT Kharagpur

Abstract. This write-up is based on an analytical comparison among 5 different Multivariate Algorithms, namely BDTs, KNN, FDA, ANNs, and SVMs, when used for classification. The algorithms are ran in TMVA[1], the ROOT library that provides the interfaces and implementations of the above mentioned machine learning techniques. A dataset from Fermilab's MiniBooNE[2] experiment has been used to accomplish this. Boosted Decision Trees emerge as the superior algorithm among all because of its high purity output and fast computation speed. The KNN model ends up as a close second.

1 Introduction

Particle identification is the process of determining the type of particle by using information left behind after it passes through a particle detector. Particle identification reduces background noise and improves measurement resolution, and it is required for many analyses at particle detectors. Various classical methods have been studied and implemented to identify different types of particles over the course of time.[3]

But, despite the presence of proven classical methods, machine learning has taken hold of a big part of particle physics research, and with good reason.

The experiments are getting bigger in scale with time and due to the proliferating massive data sets, intricate instrumentation, and state-of-the-art computing infrastructure, particle physics has benefited from, and in many ways enhanced and advanced, advances in MVA, AI/ML for decades. As the size of the datasets continue to increase, we will need more sophisticated computational methods that are able to handle such complexity.

2 The MiniBooNE Experiment

Fermilab's MiniBooNE[2] project uses a Cherenkov detector to track neutrino oscillations (BooNE is an acronym for the Booster Neutrino Experiment). A detector lined with 1,280 photomultiplier tubes and 800 tonnes of mineral oil (ultra-refined methylene compounds) is the target of a neutrino beam made mostly of muon neutrinos. The neutrino oscillation interpretation of the LSND (Liquid Scintillator Neutrino Detector) result would be supported by an excess of electron neutrino events in the detector.

Data collection for MiniBooNE began in 2002 and continued until 2017. The experiment's researchers revealed a potential signal in May 2018 that might point to the presence of sterile neutrinos.

Neutrino oscillations were demonstrated through experimental observation of solar and atmospheric neutrinos, indicating that neutrinos have masses. Due to incompatibility with oscillation parameters reported by other neutrino experiments within the context of the Standard Model, data from the LSND experiment at the Los Alamos National Laboratory are disputed. One of the experimental findings must have an alternative explanation, or the Standard Model must be extended. Additionally, the Karlsruhe KARMEN experiment looked at a [low energy] region similar to the LSND experiment but found no evidence of neutrino oscillations. LSND was more sensitive than this experiment, yet both results could be accurate.

Cosmological evidence, such as the ms 0.26 eV (0.44 eV) at 95 percent confidence limit provided by Dodelson et al., can offer an indirect but more model-dependent constraint to the mass of sterile neutrinos. However, models with different presumptions, like that of Gelmini et al., can handle cosmological evidence.

MiniBooNE was created to clearly confirm or deny the contentious LSND outcome in a controlled setting.

3 Problem Statement

The problem statement requires us to distinguish between two flavours of neutrinos, namely electron neutrinos (signal) from muon neutrinos (background). A dataset from Fermilab's MiniBooNE experiment has been used to accomplish this, through 5 different algorithms.

- Boosted Decision Trees (Adaboost)
- K-Nearest Neighbour
- Artificial Neural Networks
- Support Vector Machines
- Fischer Discriminant

As for the structure of the dataset is concerned, 36,499 signal events come first, followed by 93,565 background events, making a total of 1,30,064 instances. There are 50 particle ID variables (real) for each event.

4 Data Preparation

The algorithms were ran on 50 percent of the total data available. In other words, the usable dataset contained 18,250 events each for signal and 46,783 events for background, a total of 65,033 instances.

For training, 60,033 events, split into 15,750 signal events and 44,283 background events were used. And 5,000 events (equal split between signal and background) were used for testing.

The correlation matrices between the 50 variables are displayed in figures 1 and 2.

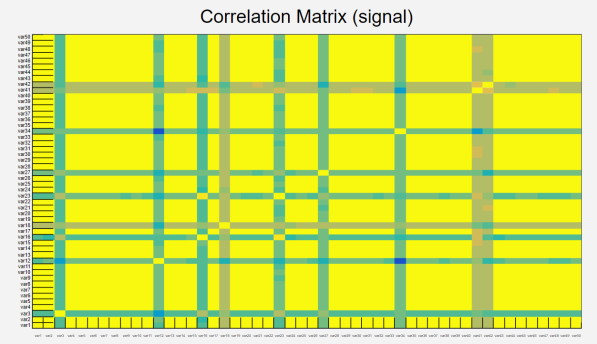

Figure 1: Correlation matrix for signal data

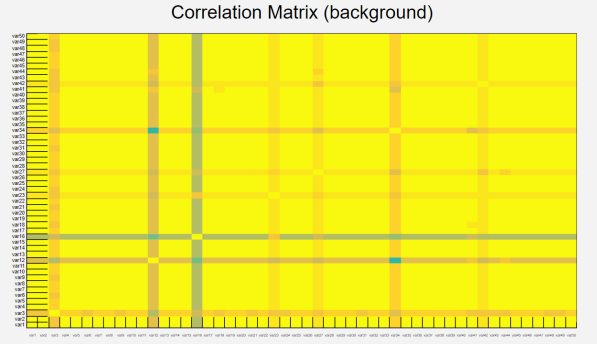

Figure 2: Correlation matrix for background data

5 The Algorithms and Their Corresponding Results

5.1 Boosted Decision Trees (Adaboost)

AdaBoost, short for Adaptive Boosting, is a statistical classification meta-algorithm formulated by Yoav Freund and Robert Schapire in 1995, who won the 2003 Gödel Prize for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. The outputs of the other learning algorithms, or "weak learners", are merged to create a weighted total that represents the boosted classifier's final results. Although AdaBoost can be used to many classes or bounded intervals on the real line, it is often used for binary classification.

AdaBoost is adaptive in the sense that it modifies future weak learners in favour of instances that prior classifiers incorrectly classified. It may be less prone to the overfitting issue than other learning algorithms in particular situations. It can be demonstrated that the final model converges to a strong learner even if the performance of each individual learner is

just marginally better than random guessing[1].

It is frequently used to combine weak base learners (like decision stumps), but it has been demonstrated that it can also combine strong base learners (like deep decision trees) well, leading to an even more precise model.

Every learning algorithm has different parameters and configurations that must be adjusted before it performs at its best on a dataset. Some learning algorithms are more suited to certain issue types than others. The best out-of-the-box classifier is frequently referred to as AdaBoost (with decision trees as the weak learners). When combined with decision tree learning, data from each stage of the AdaBoost algorithm on the relative "hardness" of each training sample is given into the method for developing the tree, causing later trees to concentrate on examples that are more difficult to categorise.

5.1.1 Training

The term "AdaBoost" designates a specific approach of boosting a classifier. A classifier of the form "boosted classifier"

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

where each f_t accepts an object x as input and returns a value reflecting the item's class. As an illustration, in the two-class problem, the predicted object class is identified by the sign of the weak learner's output, and the confidence in that classification is shown by the absolute value. The Tth classifier behaves similarly, being positive when the sample belongs to a positive class and negative otherwise.

For each sample in the training set, each weak learner generates an output hypothesis $h()$ that fixes a prediction $h(x_i)$. A weak learner is chosen at each iteration t and given a coefficient α_t so that the overall training error E_t of the resulting $t-$stage boosted classifier is minimised.

$$E_t = \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)]$$

$F_{t-1}(x)$ is the boosted classifier that has been developed up to the previous training stage, $E(F)$ is some error function, and $f_t(x) = \alpha_t h(x)$ is the weak learner that is being considered for inclusion in the final classifier.

Each sample in the training set is given a weight $w_{i,t}$ that is equal to the current error $E(F_{t-1}(x_i))$ on that sample at the end of each iteration of the training process. The weak learner can be trained using these weights, for example, by growing decision trees that prefer dividing sets of data with high weights.

Algorithm .1 *AdaBoost*

1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

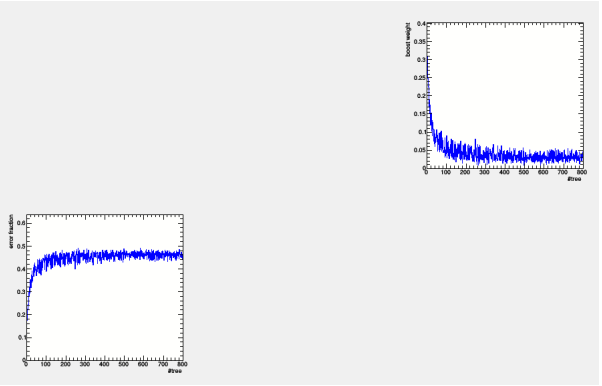

Figure 3: Control plots for BDT: Plots distributions of boost weights throughout forest, boost weights versus decision tree and error fraction.

5.1.2 Configuration

For the problem, we have used 800 decision trees, and Adaboost as the boosting algorithm. Each decision tree has a maximum depth of 3

and gini index is used as the split determination function. Also, each variable in the dataset underwent normalisation in the preprocessing stage for easy training. All of the remaining parameters were kept the same as default in TMVA.

5.1.3 Results

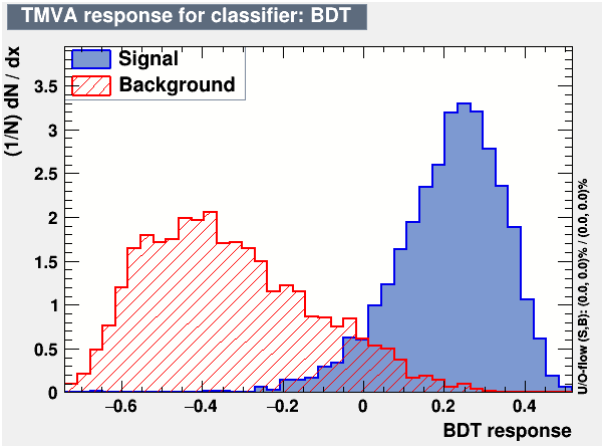

Figure 4: Classifier output distribution plots for BDT

ET for training with 60033 events: 79.8 sec
ET for evaluation of 60033 events: 2.61 sec
ET for evaluation of 5000 test events: 0.165 sec

Significance $(\frac{S}{\sqrt{S+B}}) = 29.4648$
Purity $(\frac{S}{S+B}) = 0.911$ (91.1 percent)

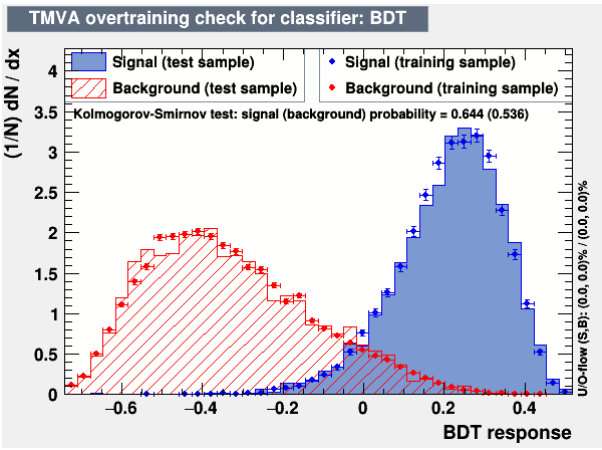

Figure 5: K-S Test for BDT

5.2 K-Nearest Neighbor

The k-nearest neighbours algorithm (k-NN) in statistics was created by Evelyn Fix and Joseph Hodges in 1951 and later improved by Thomas Cover. It is a non-parametric

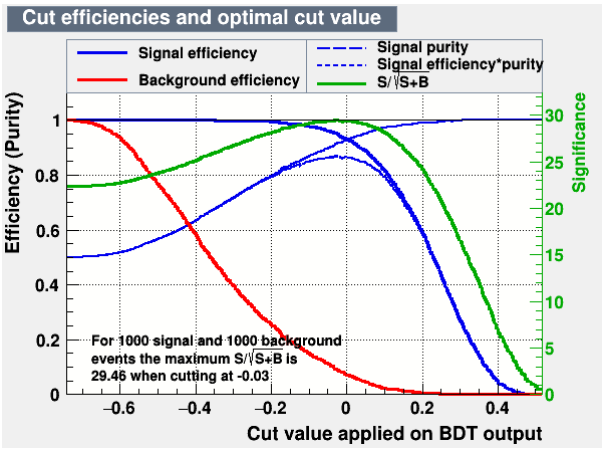

Figure 6: Cut efficiencies and optimal cut value for BDT

supervised learning technique.

It is utilised both for regression and classification. Both times, a data set's k closest training examples serve as the input. Whether k-NN is applied for regression or classification determines the results:

- In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

- In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

With k-NN classifiers, all computation is postponed until after the function has been evaluated and the function is only locally approximated. Since this technique relies on distance for classification, normalising the training data can significantly increase accuracy if the features reflect several physical units or have distinct sizes[1].

5.2.1 Configuration

For the problem, the number of k-nearest neighbors we have considered is 20. The fraction of events used to compute variable width is taken to be 0.8. Also, equal number of signal and background events have been used to train the model.

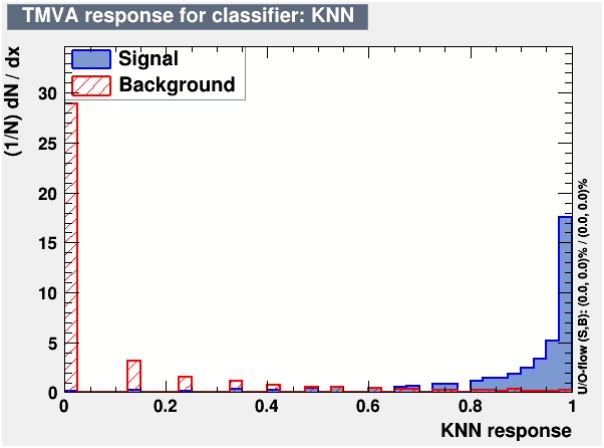

Figure 7: Classifier output distribution plots for KNN

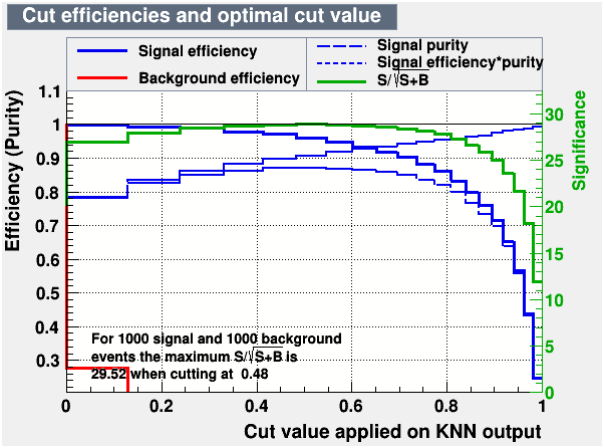

Figure 9: Cut efficiencies and optimal cut value for KNN

5.2.2 Results

ET for training with 60033 events: 0.553 sec
ET for evaluation of 60033 events: 455 sec
ET for evaluation of 5000 test events: 37 sec

Significance $(\frac{S}{\sqrt{S+B}}) = 29.5152$
Purity $(\frac{S}{S+B}) = 0.907$ (90.7 percent)

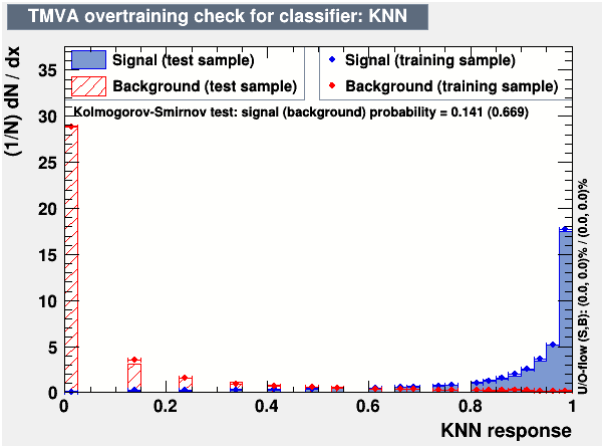

Figure 8: K-S Test for KNN

5.3 Artificial Neural Networks

Artificial neurons, which are a set of interconnected units or nodes that loosely resemble the neurons in a biological brain, are the foundation of an ANN. Like the synapses in a human brain, each link has the ability to send a signal to neighbouring neurons. An artificial neuron can signal neurons that are connected to it after processing signals that are sent to it. The output of each neuron is calculated by some non-linear function of the sum of its inputs, and the "signal" at a connection is a real number. Edges refer to the connections. The weight of neurons and edges often changes as learning progresses. The weight alters a connection's signal intensity by increasing or decreasing it. Neurons may have a threshold, and only send a signal if the combined signal crosses it. Neurons frequently group together into layers. Different layers may modify their inputs in different ways. Signals move through the layers, perhaps more than once, from the first layer (the input layer) to the last layer (the output layer)[1].

5.3.1 Training

When processing samples that each have a known "input" and "output," neural networks learn (or are trained) by creating probability-weighted associations between the two that are then stored within the net's data structure. In order to train a neural network from a given example, one often compares the processed output of the network—often a prediction—against the desired output. The error is in this discrepancy. The network then modifies its weighted associations using this error value and a learning strategy. The neural network will produce output that is increasingly comparable to the goal output as modifications are made over time. These modifications can be made a sufficient number of times before the training can be stopped under certain conditions[1]. This is known as supervised learning.

These systems typically do not have task-specific rules written into them; instead, they "learn" to do tasks by taking into account instances.

5.3.2 Configuration

Tanh is used as the activation function for the neurons. 500 training cycles are considered. Backpropagation has been used to train the model. The learning rate is taken to be 0.02 and the decay rate is taken to be 0.01. Also, Each variable in the dataset underwent normalisation in the preprocessing stage for easy training.

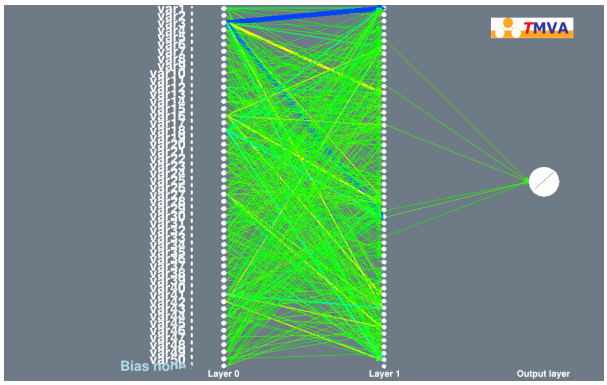

Figure 10: Network architecture

5.3.3 Results

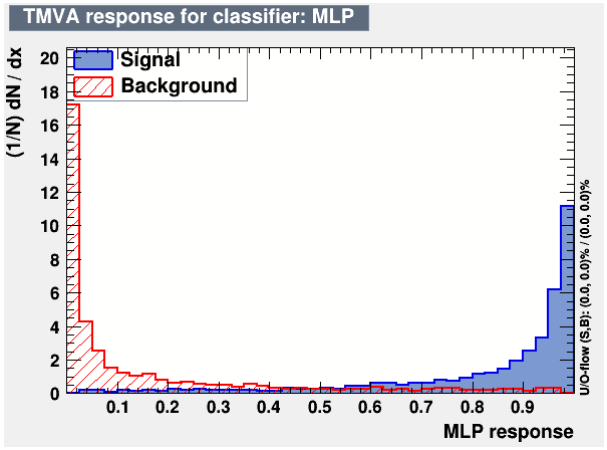

Figure 11: Classifier output distribution plots for ANN

ET for training with 60033 events: 979 sec
ET for evaluation of 60033 events: 0.692 sec
ET for evaluation of 5000 test events: 0.059 sec

Significance $(\frac{S}{\sqrt{S+B}}) = 28.2335$
Purity $(\frac{S}{S+B}) = 0.868$ (86.8 percent)

5.4 Support Vector Machine

Support-vector machines (SVMs, also known as support-vector networks) in machine learning are supervised learning models with corresponding learning algorithms that evaluate

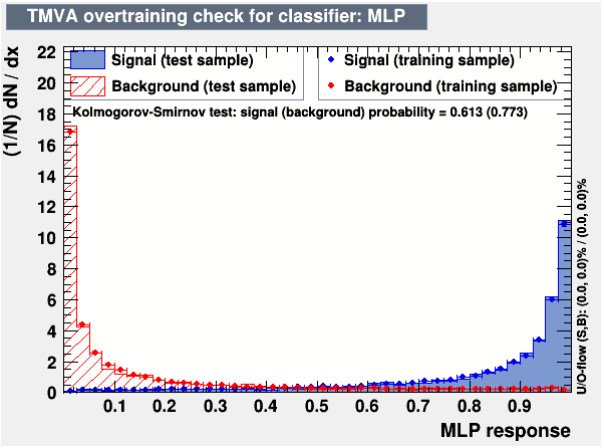

Figure 12: K-S Test for ANN

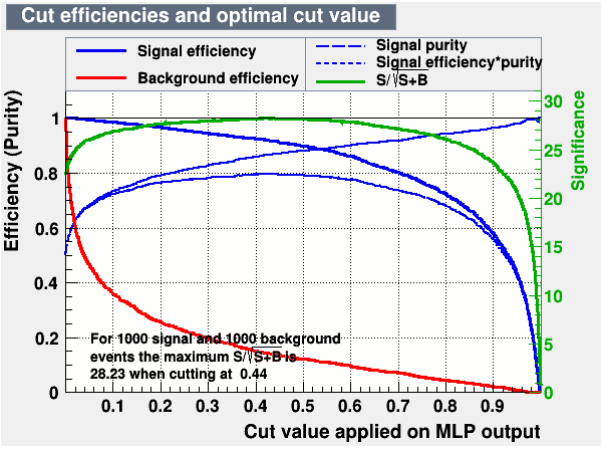

Figure 13: Cut efficiencies and optimal cut value for ANN

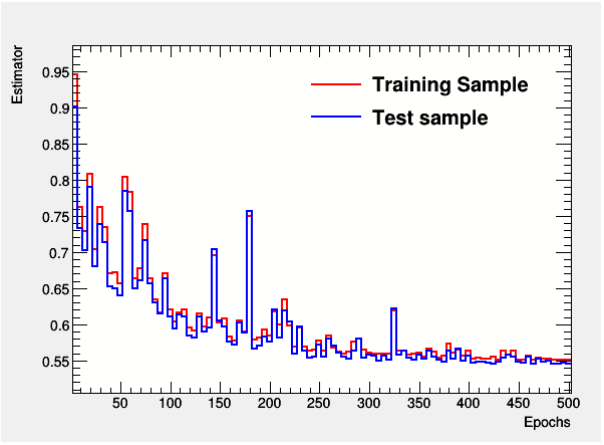

Figure 14: Convergence plot for ANN

data for regression and classification. Developed by Vladimir Vapnik and colleagues at ATT Bell Laboratories (Boser et al., 1992, Guyon et al., 1993, Cortes and Vapnik, 1995, Vapnik et al., 1997) SVMs, which are based on statistical learning frameworks or the VC theory put out by Chervonenkis and Vapnik (1982, 1995), are among the most reliable

prediction techniques (1974).

An SVM training algorithm creates a model that categorises fresh samples to one of two categories based on a collection of training examples, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). SVM assigns training samples to spatial coordinates in order to maximise the distance between the two categories. Then, based on which side of the gap they fall, new samples are projected into that same area and predicted to belong to a category[1].

Using a technique known as the kernel trick, SVMs can effectively do non-linear classification in addition to linear classification by implicitly mapping their inputs into high-dimensional feature spaces.

5.4.1 Configuration

The gamma parameter is taken to be 0.25. Tolerance set at 0.001, and the maximum number of training loops at 1000. Also, Each variable in the dataset underwent normalisation in the preprocessing stage for easy training.

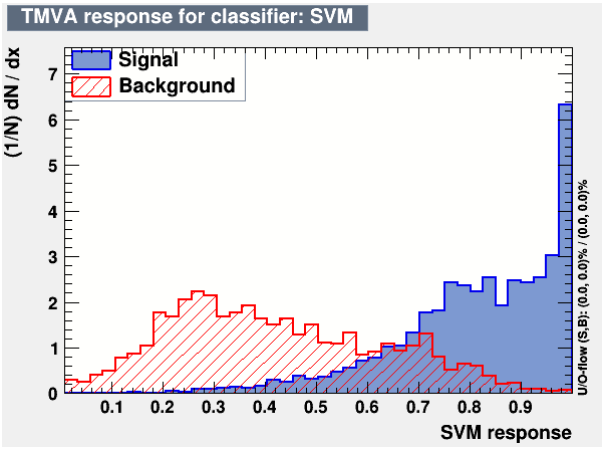

Figure 15: Classifier output distribution plots for SVM

5.4.2 Results

ET for training with 60033 events: 1.64e+03 sec
ET for evaluation of 60033 events: 135 sec
ET for evaluation of 5000 test events: 8.05 sec

Significance ($\frac{S}{\sqrt{S+B}}$) = 26.8143

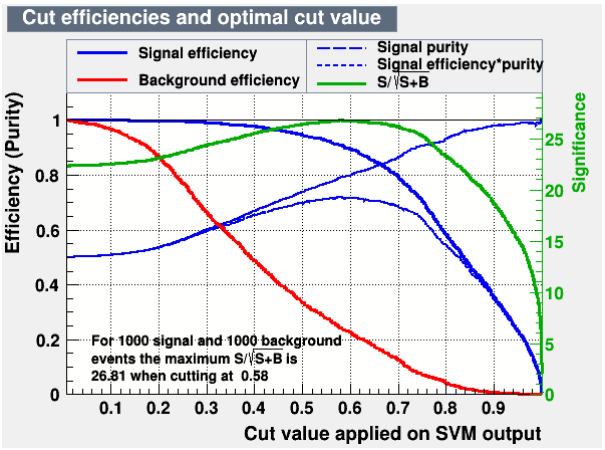

Figure 16: Cut efficiencies and optimal cut value for SVM

Purity ($\frac{S}{S+B}$) = 0.788 (78.8 percent)

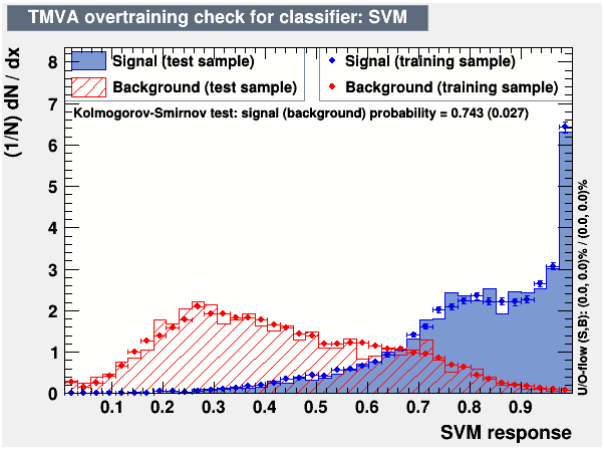

Figure 17: K-S Test for SVM

5.5 Fisher's Discriminant Analysis

Fisher's linear discriminant is a technique used in statistics and other fields to identify a linear combination of features that distinguishes between two or more classes of objects or events. Its generalisations include linear discriminant analysis (LDA), normal discriminant analysis (NDA), and discriminant function analysis. The resulting mixture can be applied as a linear classifier or, more frequently, to reduce the dimensionality prior to a subsequent classification.

Regression analysis and analysis of variance (ANOVA), which both aim to express one dependent variable as a linear mixture of other traits or measures, are closely connected to LDA. Discriminant analysis employs continuous independent variables and a categorical

dependent variable, whereas ANOVA uses categorical independent variables and a continuous dependent variable (i.e. the class label).

Because techniques similarly use the values of continuous independent variables to explain a categorical variable, logistic regression and probit regression are more comparable to LDA than ANOVA. In applications where it is not fair to assume that the independent variables are normally distributed, which is a fundamental assumption of the LDA approach, these other methods are preferred.

LDA and principal component analysis (PCA) and factor analysis are linked in that they both seek out linear combinations of variables that provide the most comprehensive explanation of the data. LDA makes a clear effort to model the distinction between the data classes. PCA, in contrast, ignores any class distinction, and component analysis constructs the feature combinations based on distinctions as opposed to similarities.

In addition to not being an interdependence technique, discriminant analysis differs from factor analysis in that it requires a differentiation between independent variables and dependent variables (also known as criterion variables). When measurements on independent variables for each observation have continuous values, LDA is effective. Discriminant correspondence analysis is the analogous method when dealing with categorical independent variables.

When groups are known a priori, discriminant analysis is utilised (unlike in cluster analysis). Each instance must have a score on a group measure as well as a score on one or more quantitative predictor measures. Discriminant function analysis can be defined as the classification of entities into groups, classes, or categories of the same kind[1].

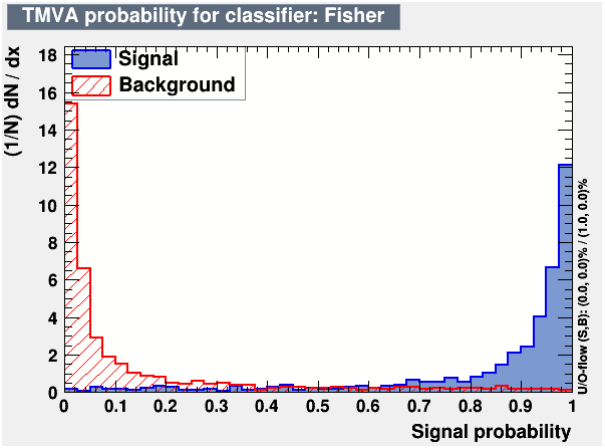

Figure 18: Probability plot for FDA

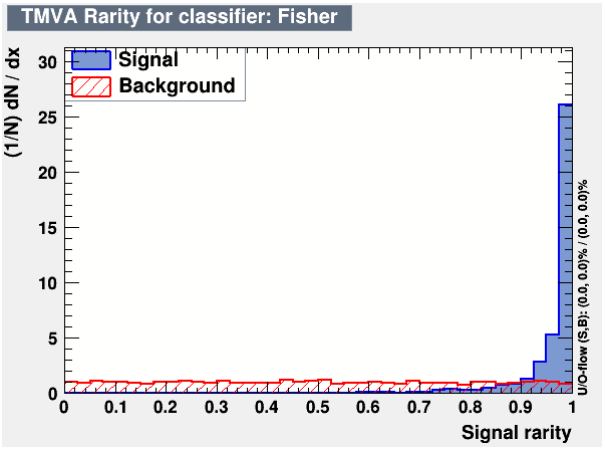

Figure 19: Plot for classifier probability integral transformation distributions for signal and background events

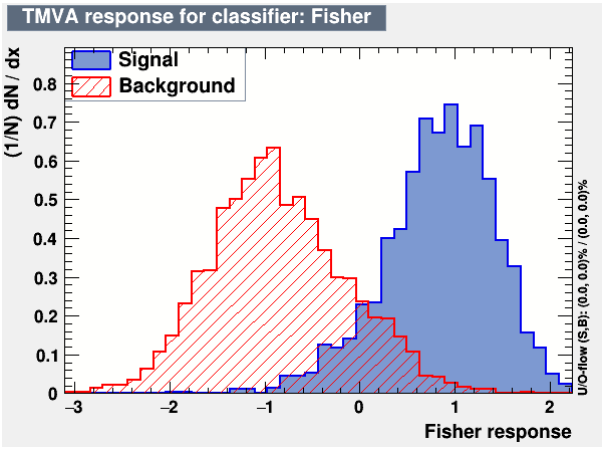

Figure 20: Classifier output distribution plots for FDA

5.5.1 Configuration

Gaussian transformation is performed on the variables in the preprocessing stage for easy training.

5.5.2 Results

ET for training with 60033 events: 2.75 sec sec
ET for evaluation of 60033 events: 0.733 sec
ET for evaluation of 5000 test events: 0.05 sec

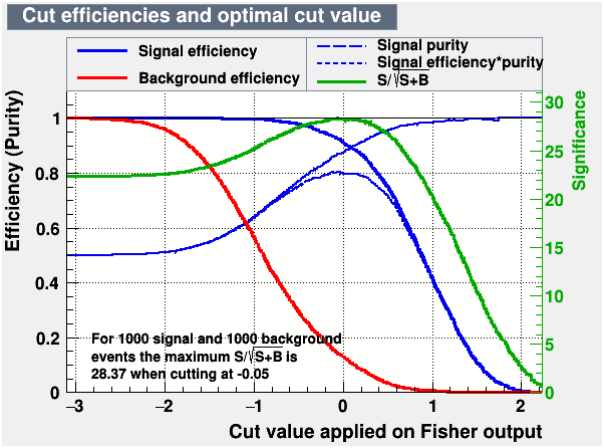

Figure 21: Cut efficiencies and optimal cut value for FDA

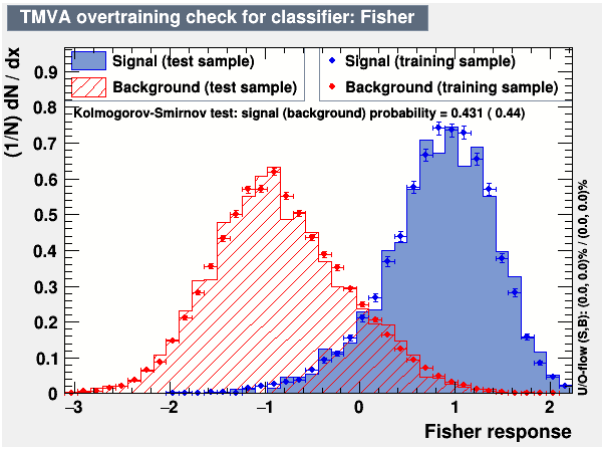

Figure 22: K-S Test for FDA

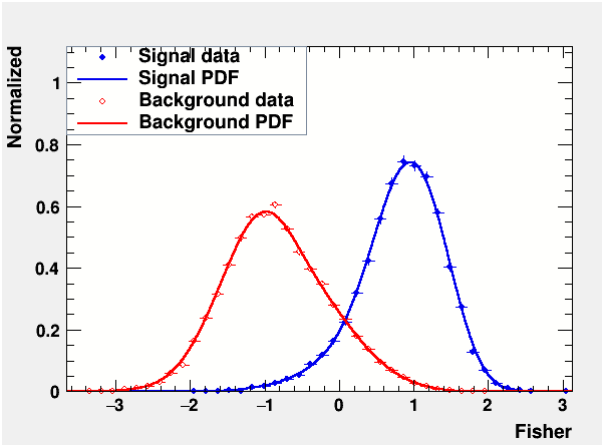

Figure 23: PDF Output response for FDA

Significance $(\frac{S}{\sqrt{S+B}}) = 28.3654$
Purity $(\frac{S}{S+B}) = 0.869$ (86.9 percent)

6 Overall Comparison

As from the tables we can understand, the BDT and KNN models outperform the others

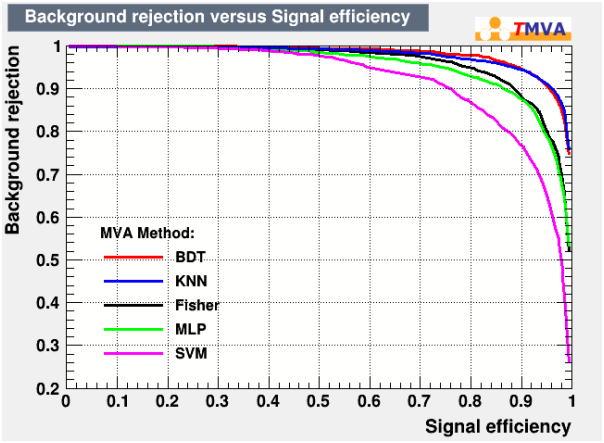

Figure 24: Plot for the background rejection versus signal efficiency ("ROC curve") obtained by cutting on the classifier outputs for the events of the test sample

Table 1: Evaluation results ranked by best signal efficiency and the ROC-Integral

Algorithm	ROC-Integral
BDT	0.981
KNN	0.978
Fisher	0.960
MLP	0.954
SVM	0.920

Table 2: Testing efficiency compared to training efficiency (overtraining check)

Algorithm	Efficiency: from test (from training)
BDT	0.955 (0.960)
KNN	0.961 (0.962)
Fisher	0.887 (0.888)
MLP	0.866 (0.872)
SVM	0.759 (0.752)

Table 3: Evaluation results ranked by significance and purity

Classifier	Significance	Effsig	Effbkg	Purity
BDT	29.4648	0.9524	0.0924	0.911
KNN	29.5152	0.9604	0.0984	0.907
Fisher	28.3654	0.9256	0.1392	0.869
MLP	28.2335	0.918	0.1392	0.868
SVM	26.8143	0.912	0.2448	0.788

by a considerable margin. Among these two, BDT provides better (marginally) purity of 91.1 percent whereas KNN provides better (marginally) signal efficiency of 0.9604 and significance of 29.5152.

Comparing the computation time required to run these two models, the BDT emerges as a superior algorithm because of its quicker total runtime of 82.575 seconds in comparison to KNN's 492.553 seconds.

Acknowledgement

This project was done under the guidance of Dr. Anand Kumar Dubey, Variable Energy Cyclotron Centre, Kolkata.

References

[1] Hoecker, A., Speckmayer, P., Stelzer, J., Therhaag, J., Toerne, E. von, and Voss, H. "TMVA: Toolkit for Multivariate Data Analysis". In: *PoS* ACAT (2007), p. 040. arXiv: physics/0703039.

[2] Ray, H. "The MiniBooNE Experiment". In: *SLAC Summer Institute On Particle Physics* ACAT (2006), p. 006.

[3] Tully, C. G. "Elementary particle physics in a nutshell". In: ().