Hands-on Tree-based Algorithms Training Technical Note from Scratch (in Python)

Shaokai Yang[1], Alex Sousa[1], and Enhao Song[2, 3]

[1]Department of Physics, University of Cincinnati
[2]Department of Physics, University of Virginia
[3]Google

CONTENTS

List of Figures

List of Tables

Hands-on Tree-based Algorithms Training Technical Note from Scratch (in Python)

***Abstract*—The High Energy Physics (HEP) community has been developing dedicated solutions for processing experiment data over decades. However, with recent advancements in Big Data, Automated Machine Learning, GPU acceleration, and Cloud (CPU & GPU) Services, a question of application of such technologies in the domain of HEP data analysis becomes relevant. As the first part of our trilogy of the Alchemists Handbook, we present our experience with tree-based algorithms training that combines the data extraction, feature engineering, automated learning, and data visualization based on the off-the-shelf frameworks, services, and tools. The exploratory data analysis (EDA) present in this technote is based on the CERN Open Data, but it can be decoupled entirely from the experiment, or in other words, it can be coupled/employed with any other HEP experiments. We also provide an interactive web-based interface based on Jupiter Notebooks as the main entry-point for the potential users.**

***Keywords*—Random Forest, Boosted Decision Trees, Automated Machine Learning Platform, Tree-Based Pipeline Optimization(TPOT), Jupiter Notebooks.**

I. Introduction

Decision Trees (DT) are comprehensive, predictive modeling tools that have applications spanning various branches of HEP, such as rejecting cosmic backgrounds in NOvA [2] and discovering Higgs Boson in LHC [3]. Besides, various machine learning algorithms can be trained through ensemble learning methods into a single complex model. This integrating approach allows the production of better predictive performance and the increase of the stability concerning statistical fluctuations in the training set compared to a sole model. Hence, how to effectively train DTs by ensemble learning methods are useful and exciting skills for HEP data analyst. In this technote, we start by introducing decision tree algorithms from scratch and presenting a series of current methods for constructing a decision tree in a top-down manner, including different splitting criteria and pruning methodology. Then, we give a concise introduction to the ensemble methods which can combine homogeneous or heterogeneous decision trees into one tree-based complex model in order to decrease variance (bagging) or bias (boosting). Based on the distinct ensemble methods, we then fully comprehensive present different kinds of Random Forest (RF) and Boosted Decision Trees (BDT) separately. After described the state-of-the-art tree-based models, we illustrate how to employ (tree-based) automated machine learning frameworks to tune model hyper-parameters, engineer input features, and select suitable algorithms based on your data. Finally, in the appendix part, we give detailed examples of how to perform the training phase for the above-described algorithms and tools in python based on simulated (signal) and measured (backgrounds) NOvA Far Detector (FD) data.

II. Decision Trees (DT)

DT is a white box type of machine learning (ML) algorithms. It shares inner decision-making logic with the data analyst, which is not available in the black box type of algorithms, such as the convolutional neural networks (CNN), until now. General speaking, a trained DT model is created by an algorithmic approach which identifies ways to split a full data set into subgroups based on various conditions. It is known as the easiest to interpret and practical algorithm for supervised learning. Also, it can be employed for both classification and regression problems. The aim is to train a tree-like graph or model that predicts the value of a target by learning decision rules inferred from the input training (data) set. The decision rules are in the form of if-then-else statements.

Before we dive deep, let us first get familiar with some of the tree-based algorithm related vocabulary and jargon:

Feature: A feature (also known as attribute) is a quantity describing an input data or instance;
Instance: An instance is an input data, which is the vector of features or attributes that define the input space;
Training Set: A set of input data paired with a label, which is the correct output;
Testing Set: Similar to the training set and is used to test the trained model and determine its performance;
Root Node: It represents entire training set and this further gets divided into two or more homogeneous sets;
Splitting: It is a process of dividing a node into two or more sub-nodes;
Decision Node: When a sub-node splits into further sub-nodes, then it is called a decision node;
Leaf: Nodes do not split is called Leaf or Terminal node;
Pruning: When we remove sub-nodes of a decision tree, this process is called pruning. It is the opposite process of splitting;
Branch: A sub section of entire tree is called branch or sub-tree;
Child Node: A node, which is divided into sub-nodes is known as parent node of the sub-nodes where as sub-nodes are the child of parent node;
Trained Model: The function that maps input to output.

A. Tree Structure

A trained decision tree model can be explained by a tree-like graph, as shown in the Fig1, where an internal node stands for

one feature of the input data, the branch stands for a decision rule, and each leaf node stands for a type of target.

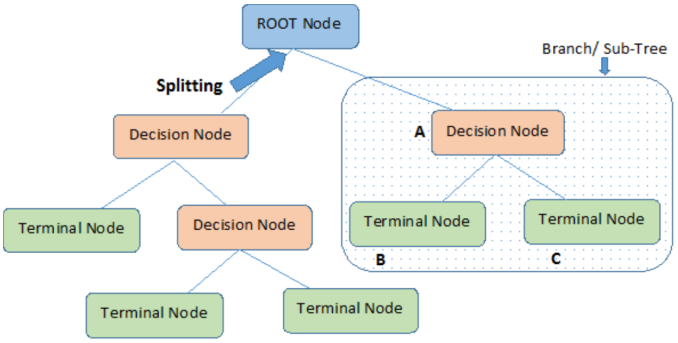

Fig. 1. A general decision tree structure, which illustrates all the fundamental elements of a tree, such as Root Node, Decision Note, Terminal Node/Leave, and Branch/Sub-Tree. Credit by Analytics Vidhya Content Team

This algorithm learns to partition based on the input data features. It partitions the decision tree in a recursive manner named recursive partitioning. Its visualization looks like a flowchart diagram. This flowchart-like structure mimics the human-level thinking in making decision. That is why DT algorithms are easy to understand and interpret.

Following the above discussion of the core concepts of the algorithm, let us outline a specific application to the NOvA particle interactions classification. NOvA makes use of the Neutrinos at the Main Injector (NuMI) neutrino beam to measure ν_μ to ν_e, $\bar{\nu_\mu}$ to $\bar{\nu_e}$, and active-sterile neutrino oscillation by comparing the observed Charged-Current (CC) and Neutral-Current (NC) neutrino-nuclear interactions in two detectors separated by 810 km. A core problem in NOvA, more generally in HEP experiments, is the correct categorization of the particle interactions recorded in the detector(s) as signal and backgrounds. The NOvA FD data is used for the training and evaluation of tree-based algorithms and tools in this technote. The NOvA FD, can be seen in Fig2, is composed of extruded PVC cells (15.5 m long) filled with liquid scintillator which divide the detector into cells with a cross-section 3.9 cm (wide) $\times$ 6.6 cm (deep). Scintillation light, caused by moving charged particles, will be captured by a wavelength shifting fiber which permeates each cell in NOvA FD. The end of the fiber is accessed to a single pixel on an avalanche photo-diode (APD) arrange to record the intensity and arrival time of photon signals. The spatial and absolute response of the NOvA FD to deposited light is then calibrated out using physical standard candles to make sure that a final calibrated response can be derived which is a good enough estimate of the true deposited energy. Parallel cells are arranged into planes, which are configured in alternating horizontal and vertical alignments to provide detached, interleaved X-Z (top), and Y-Z (side) views. The 14,000 ton FD consists of 344,064 total channels arrayed into 896 planes each 384 cells wide [1]. Information from these two views can be combined to allow three dimensions (3D) of particle interaction reconstruction. The reconstructing high-level components such as clusters, tracks, and showers associated with particle interactions recorded by the detector and summarizing the energies, directions, and shapes of these objects can be selected as the input data features for building our classification tree. Furthermore, NOvA CVN group [4] developed a CNN-based neutrino interactions identifier, which is based on their topology without the need for detailed reconstruction. We also employ the CVN output as one of the input features for decision tree algorithm training.

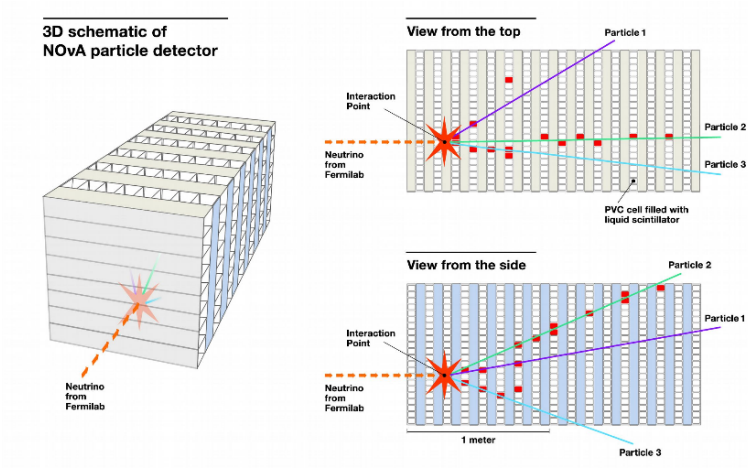

Fig. 2. The right side two figures show the top and side views of the 3D figure on the left. They show the 'hits' produced as charged particles pass through and deposit energy in the scintillator-filled cells. Illustration courtesy of Fermilab.

Let's assume we want to classify NC and Cosmic events (interactions) recorded by NOvA FD based on the following three features:

1) CVN Cosmic ID: A CNN-based variable. The higher the value, the more the probability of being a cosmic ray caused interaction;

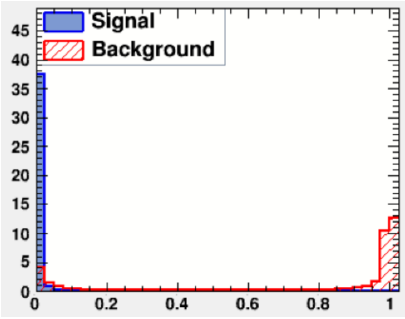

Fig. 3. FD NC (Signal) vs Cosmic (Background) CVN Cosmic ID distribution plot. Based on this plot, we can see that the CVN Cosmic ID gives us the most robust discrimination ability among the 3 input features (event-variables), and therefore it has been employed as the root splitting variable in our sample Decision Tree model.

2) Leading Shower (LS) Energy: The longest shower's reconstructed energy;

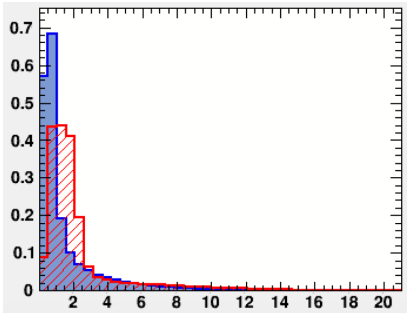

Fig. 4. FD NC (Signal) vs Cosmic (Background) Leading Shower Energy distribution plot. Based on this plot, we can see that the LS Energy gives us the weaker discrimination ability compare to the Cosmic CVN ID. (The Blue plot is NC and the red one presents Cosmic)

3) Number of FuzzyK Prongs: Number of the reconstructed prongs, which are a collection of hits with a reconstructed path through them.

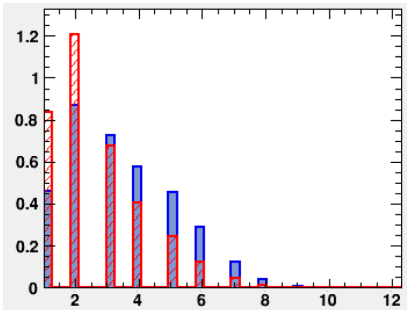

Fig. 5. FD NC (Signal) vs Cosmic (Background) Number of FuzzyK Prongs distribution plot. Based on this plot, we can see that the Number of FuzzyK Prongs gives us the weakest discrimination ability among the 3 input features (event-variables). (The Blue plot is NC and the red one presents Cosmic)

We use 1,000 NC and 1,000 Cosmic events to train a single decision tree prediction model as present in Fig6.

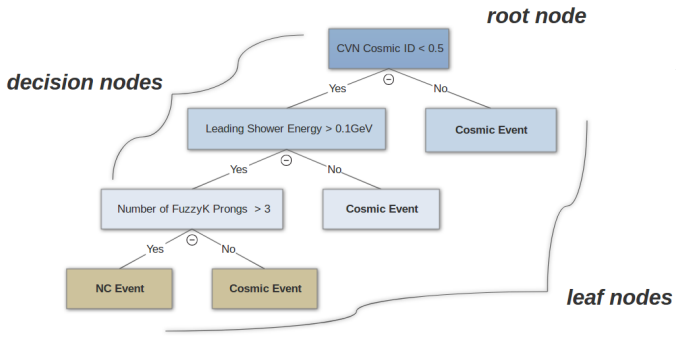

Fig. 6. This simple sample decision tree model was trained with NOvA FD simulated (NC) events and measured (Cosmic) events.

Let's illustrate the trained tree categorization way with help of the following selected events as listed in table I.

TABLE I. SELECTED NC AND COSMIC EVENTS

Event	Cosmic CVN ID	LS Energy	Prong Number	Label
1	2.54555e-06	2.64721	5	NC
2	3.11355e-05	0.503545	4	NC
3	0.000195934	0.478824	4	NC
4	0.559242	0.13050	2	NC
5	0.250129	4.99891	3	NC
6	0.825315	1.89396	2	Cosmic
7	0.954139	0.356619	1	Cosmic
8	0.662547	0.0759066	1	Cosmic
9	0.297158	0.863281	1	Cosmic
10	0.622075	2.57035	3	Cosmic

This model classifies the ten events by sorting them down from the root node to the four leaf nodes, with the leaf nodes labeling the NC and Cosmic separately. Each decision node in this tree behaves as a test case for one feature, and each edge descending from that node corresponds to one of the possible answers to the test case. This process is recursive and is repeated for every sub-tree rooted at the new nodes. Following this way, we can get the tree model prediction and compare them with the event labels. The results are presented in the table II.

TABLE II. TREE PREDICTED RESULTS

Event	Label	Prediction	Result
1	NC	NC	Right
2	NC	NC	Right
3	NC	NC	Right
4	NC	Cosmic	Wrong
5	NC	Cosmic	Wrong
6	Cosmic	Cosmic	Right
7	Cosmic	Cosmic	Right
8	Cosmic	Cosmic	Right
9	Cosmic	Cosmic	Right
10	Cosmic	Cosmic	Right

After a simple calculation, we can see that the tree prediction correction rate is 80%, a not so bad result. In the next section, we will detail how the tree was grown and how to improve prediction performance.

B. Tree Growing

The tree growing process is presented in Fig7. The basic concept behind the process is as follows:

1) Choosing the best feature by Feature Selection Methods (FSM);

2) Making that feature a decision node and splitting the whole training dataset into subgroups;

3) Growing the tree by repeating the above two steps recursively for each child node until no split gains sufficient splitting measure or a stopping criteria is satisfied;

4) Pruning some of the sub-trees if necessary.

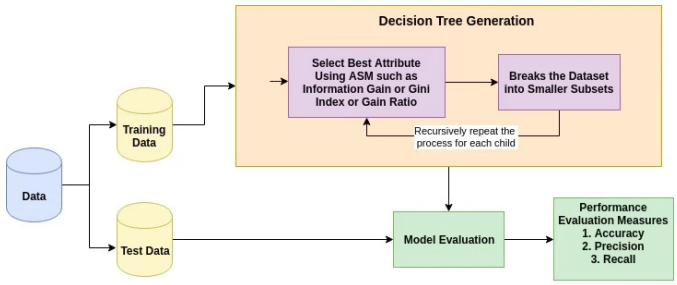

Fig. 7. Decision Tree Generation Mechanism Schematic. Credit by Avinash Navlani.

How to grow an optimal decision tree from a given dataset is considered to be a hard task. Decision-makers, like us, prefer fewer complex models, due to they are considered more comprehensible. Moreover, the tree complexity is believed as a significant effect on prediction accuracy. In one word, we want a simply but efficiency model. Let us see how the goal can be performed.

1) Splitting Criteria: Decision trees may use multiple criteria to decide to split a node into two or more sub-nodes. The creation of the children node increases the homogeneity of the resultant children node. In other words, the purity of the parent node enhances with respect to the target variable. The algorithm splits the nodes on all available variables and then selects the split, which results in most homogeneous children node. Roughly speaking, there are two types of splitting criteria: univariate splitting and multivariate splitting. In most of the cases, we employ the univariate splitting criteria, which means that one decision node is divided according to the value of a single feature. On the other hand, the multivariate splitting criteria employs several attributes to participate in a single node split test. In this technote, we will only focus on the univariate splitting criteria, which can be characterized in different ways, such as:

- On the basis of the measure structure: impurity-based criteria and binary criteria.
- On the basis of the origin of the measure: information theory, dependence, and distance;

Let us go through some of the splitting criteria which are widely used on Kaggle Competition.

Impurity-based Criteria: The goodness-of-split due to one feature is defined as the reduction in the impurity of the training dataset. The dataset impurity can be measured in various ways, therefore different criteria were invented.

Information Gain (IG): It introduces entropy (from information theory) as the impurity measure. Suppose that we have a dataset which includes totally S events and M types of classes. For each type of events, it has S_m events. Then, the entropy can be defined as:

$$Entropy = \sum_{m=1}^{M} -\frac{S_m}{S} log_2(\frac{S_m}{S}) \tag{1}$$

It needs two steps to calculate entropy for one split: 1) the parent node entropy calculation, which is employed as the weight for child node entropy calculation and 2) computing entropy of each individual node of split and calculating weighted average of all children nodes available in split. The IG then can be defined as $1 - Entropy$.

Gini Index (GI): It measures the divergences between the probability distributions of the target feature's values.

1) It just performs binary splits;
2) Higher the value of Gini Index, higher the homogeneity of the dataset.

$$GI = \sum_{m=1}^{M} (\frac{S_m}{S})^2 \tag{2}$$

Gain Ratio (GR): The gain ratio "normalizes" the information gain as follows:

$$GR = \frac{IG}{Entropy} \tag{3}$$

2) Stopping Criteria: The tree complexity is controlled by the stopping criteria and the pruning method. Generally, we measure the tree complexity is by one of the following metrics:

1) the total number of nodes;
2) the total number of leaf nodes,
3) tree depth;
4) number of employed features.

The growing procedure would continues until a stopping criterion is triggered, such as:

- All instances in the training set belong to a single value;
- The maximum tree depth has been reached;
- The number of instances in the terminal node is less than the minimum number of instances for parent nodes;
- If the node were split, the number of instances in one or more children node would be less than the preset minimum number of instances for children node;
- The best splitting criteria is not better than a certain threshold.

3) Pruning Methods: Employing loosely stopping criteria tends to produce large but overfitting models. On the other hand, Using tightly stopping criteria tends to generate small but under-fitting trees. To solve this dilemma, tree pruning methods were developed. The loosely stopping criteria are employed to grow the tree, and then the overfitting tree is cut back into a smaller one by removing sub-trees that are not contributing to the generalization accuracy. Various

studies have shown that pruning methods can enhance the generalization performance of a decision tree, particularly in noisy domains. There are various methods for pruning decision trees. Most of them perform top-down or bottom-up traversal of the nodes. A node will be pruned if this operation improves a specific criterion. The following parts describe the most popular techniques which have been employed in Kaggle Competition.

Cost-Complexity Pruning (CCP): Cost-complexity pruning (also known as weakest link pruning or error complexity pruning) proceeds in two stages. In the first stage, a sequence of trees T_0, T_1, . . .,T_k is built on the training data where To is the original tree before pruning and T_k,is the root tree. In the second stage, one of these trees is chosen as the pruned tree, based on its generalization error estimation.

Reduced Error Pruning (REP): A simple procedure for pruning decision trees, known as reduced error pruning, has been suggested by Quinlan (1987). While traversing over the internal nodes from the bottom to the top, the procedure checks for each internal node, whether replacing it with the most frequent class does not reduce the tree's accuracy. In this case, the node is pruned. The procedure continues until any further pruning would decrease the accuracy. In order to estimate the accuracy, Quinlan (1987) proposes to use a pruning set. It can be shown that this procedure ends with the smallest accurate sub-tree with respect to a given pruning.

Error-based Pruning (EBP): Error-based pruning is an evolution of pessimistic pruning. It is implemented in the well-known C4.5 algorithm.

C. Tree Inducers

The tree inducers are algorithms that automatically train a decision tree model from the given training dataset. Typically, the training goal is to find the optimal model by minimizing the generalization error. Other target functions can also be defined, for instance, minimizing the number of decision nodes or minimizing the average tree depth.

1) ID3 Tree: The ID3 tree is considered as a straightforward algorithm. It uses information gain as the splitting criterion. The growing phase will stop when the whole training dataset belongs to a single value of target feature or when best information gain is not greater than zero. It does not apply pruning procedures, nor does it handle numeric attributes or missing values.

2) C4.5 Tree: It is an evolution version of ID3 tree. The C4.5 tree employs the gain ratio as the splitting criterion. The splitting phase stops when the number of instances to be split below a certain threshold. Error-based pruning method is used after the growing phase. It has the ability to handle numeric features. It also can induce a training set that incorporates missing values by using the corrected gain ratio criterion.

3) Classification and Regression Trees (CART): It is characterized by the binary trees inducer, which means that each internal node has exactly two outgoing edges. The splits are selected using the two criteria, and the obtained tree is pruned by CCP method. When provided, CART can consider misclassification costs in the tree induction. It also enables users to provide prior probability distribution. A primary feature of CART is it can generate a regression tree model, which are trees where their leaves predict a real number and not a class. In the case of regression, CART looks for splits that minimize the prediction squared error (the least-squared deviation). The prediction in each leaf node is based on the weighted mean for node.

III. Ensemble Methods

The concept of ensemble methodology is to produce a predictive model by combining multiple models. It is well-known that ensemble methods can be employed for producing stronger prediction performance. In this section, we present all essential types of ensemble methods, including boosting and bagging.

A. Ensemble Learning

The basic idea of ensemble methodology is to integrate a set of trained models, each of which aims to solve the same task, in order to gain a better composite model, with enhanced accurate and reliable estimates or decisions than can be acquired from employing a single model. The idea of producing a predictive model by combining multiple models has been under investigation for a long time. In the following part, we focus on classifier ensembles, though the ensemble methods can also improve the quality and robustness of clustering and regression algorithms. Experimental studies conducted by the machine learning community show that combining the outputs of multiple classifiers reduces the generalization error. Ensemble methods are instrumental, mainly due to the phenomenon that various types of classifiers have different "inductive biases". Ensemble methods can efficaciously make use of such diversity to reduce the variance-error without increasing the bias-error. Given the potential usefulness of ensemble methods, it is not surprising that a vast number of methods is now available to HEP data analyst. We aim to organize all significant methods developed in this field into a coherent and unified catalog. Several factors differentiate between the various ensembles methods. They are:

1) Inter-models relationship – How does each trained model affect the others? The ensemble methods can be categorized into two main classes: sequential and concurrent;

2) Combining process – The strategy of combining the trained models generated by an induction algorithm;

3) Diversity generator – In order to make the ensemble efficient, there should be some diversity between the trained models. Diversity may be acquired by different

presentations of the input data, such as in bagging method, variations in learner design, or by adding a penalty to the output to encourage diversity;

4) Ensemble size – The number of trained models in the ensemble process.

For tree-based algorithm related ensemble methods, they are meta-algorithms which integrate machine learning techniques into one predictive composite model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking). These types of ensemble methods can be divided into two groups:

1) Concurrent (Parallel) methods: where the trees are generated in parallel (e.g., Random Forest). The primary motivation of these methods is to exploit independence between the trees as the error can be reduced dramatically by averaging.

2) Sequential methods: where the trees are generated sequentially (e.g., Adaptive Boosting).The motivation of these methods is to exploit the dependence between the trees. The integrated performance can be boosted by weighing previously mislabeled instances with higher weight;

B. Bagging Algorithm

Bagging (also called bootstrap aggregation) is a way to reduce the prediction variance by averaging together multiple trees. For example, we train N trees on different subsets of the dataset (chosen randomly with replacement) and calculate the ensemble:

$$f(x) = \frac{1}{N}\sum_{n=1}^{N} f_n(x) \tag{4}$$

Bagging methods employ bootstrap sampling to gain data subsets for training the trees. For aggregating the outputs of the trained trees, bagging uses voting for classification and averaging for regression.

C. Boosting Algorithm

Boosting methods refer to a family of algorithms which convert weak learners to active learners. The core idea of boosting is to fit a sequence of trees that are only slightly better than random guessing to weighted versions of the training dataset. Higher weight is given to instances that were misclassified by earlier trees. The outputs are then combined through a weighted majority vote (classification) or a weighted sum (regression) to get the final result. The significant difference between boosting and bagging is that the trees are trained in sequence on a weighted version of the data. The algorithms below describe the most widely used form of boosting algorithms, including AdaBoost, which stands for adaptive boosting, and gradient boosting.

IV. Random Forest (RF)

A. Brief Introduction of RF

RFs are a scheme proposed by Leo Breiman for producing a predictor ensemble with a series of decision trees which grow in randomly selected sub-group of the training dataset. Notwithstanding increasing interest and practical use in the HEP community, there has been a little exploration of the statistical properties of RFs, and little is known about the mathematical forces driving this type of algorithms. Fortunately, with libraries such as Scikit-Learn, it is now easy to implement hundreds of machine learning algorithms in Python. While knowing all the details is not essential, it is still helpful to have an idea of how RF algorithms work under the hood. This enables us to diagnose the trained model when it is underperforming or illustrate how the model makes decisions, which is crucial if we want to convince other colleagues to trust our results. In this section, we offer an in-depth analysis of this type of algorithms. We will look at how to build and apply the RF in Python. In addition to seeing the code, we will try to get an understanding of how this algorithm works. Then, we will work our way of employing it on a NOvA classification problem. To show respect to Breiman, we employ CART, his another contribution, to build our RF model.

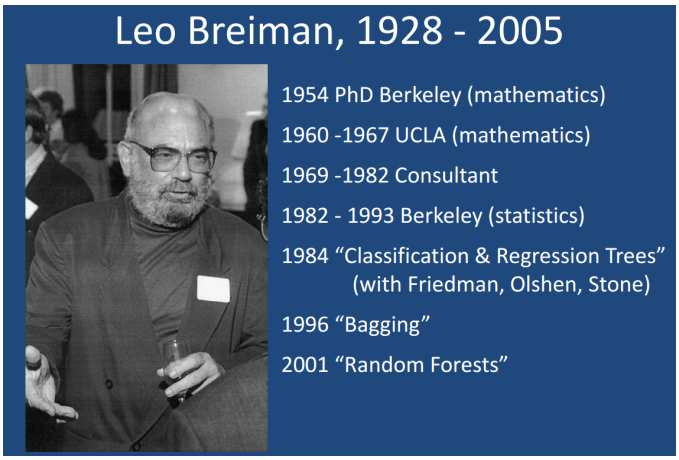

Fig. 8. Leo Breiman and Random Forests. Credit by Adele Cutler.

B. RF in Theory

But a decision tree suffers from high variance. "High Variance" means getting high prediction error on unseen data. We can overcome the variance problem by using more data for training. But since the data set available is limited to us, we can use re-sampling techniques like bagging and random forest to generate more data. In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e. a bootstrap sample) from the training set. In addition, instead of using all the features, a random subset of features is selected, further randomizing the tree. As a result, the bias of the forest increases slightly, but due to the averaging of less correlated trees, its variance decreases,

resulting in an overall better model. RF is a type of tree-based algorithm which involves building different decision trees, then combining their output to improve the generalization ability of the model. The method of combining trees is known as an ensemble method. Ensemble is nothing but a combination of weak learners (individual trees) to produce a stronger learner. For example, we want to know how good is a movie. We can ask ten people who have watched the video. If eight of them say, " the film is fantastic." Since the majority is in favor, you decide to view it. This example shows how we use ensemble techniques in our daily life. RF can be used to solve regression and classification problems. In regression problems, the dependent variable is continuous. In classification problems, the dependent variable is categorical.

C. RF in Practice

Building many decision trees results in a forest. A random forest works the following way:

1) First, it uses the Bagging (Bootstrap Aggregating) algorithm to create random samples. Given a data set D1 (n rows and p columns), it creates a new dataset (D2) by sampling n cases at random with replacement from the original data. About 1/3 of the rows from D1 are left out, known as Out of Bag(OOB) samples;
2) Then, the model trains on D2. OOB sample is used to determine unbiased estimate of the error;
3) Out of p columns, P « p columns are selected at each node in the data set. The P columns are selected at random. Usually, the default choice of P is p/3 for regression tree and P is sqrt(p) for classification tree;
4) Unlike a tree, no pruning takes place in random forest; i.e, each tree is grown fully. In decision trees, pruning is a method to avoid overfitting. Pruning means selecting a subtree that leads to the lowest test errror rate. We can use cross validation to determine the test error rate of a subtree;
5) Several trees are grown and the final prediction is obtained by averaging or voting

V. Boosted Decision Trees (BDT)

A. Boosting in general

Boosting is a sequential process; i.e., trees are grown using the information from a previously grown tree one after the other. This process slowly learns from data and tries to improve its prediction in subsequent iterations. Let's look at a classic classification example:

Let's understand this picture well:

- Box 1: The first classifier creates a vertical line (split) at D1. It says anything to the left of D1 is + and anything to the right of D1 is -. However, this classifier misclassifies three + points;
- Box 2: The next classifier says don't worry I will correct your mistakes. Therefore, it gives more weight to the three + misclassified points (see bigger size of +) and creates a vertical line at D2. Again it says, anything to right of D2 is - and left is +. Still, it makes mistakes by incorrectly classifying three - points;
- Box 3: The next classifier continues to bestow support. Again, it gives more weight to the three - misclassified points and creates a horizontal line at D3. Still, this classifier fails to classify the points (in circle) correctly. Remember that each of these classifiers has a misclassification error associated with them;
- Boxes 1,2, and 3 are weak classifiers. These classifiers will now be used to create a strong classifier Box 4;
- Box 4: It is a weighted combination of the weak classifiers. As you can see, it does good job at classifying all the points correctly.

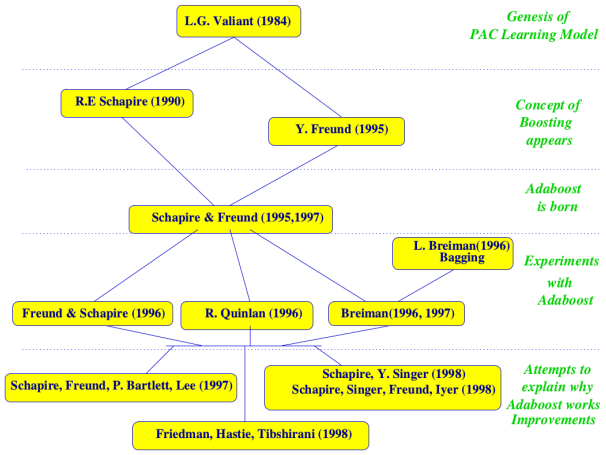

Fig. 9. Boosting History. Credit by Trevor Hastie

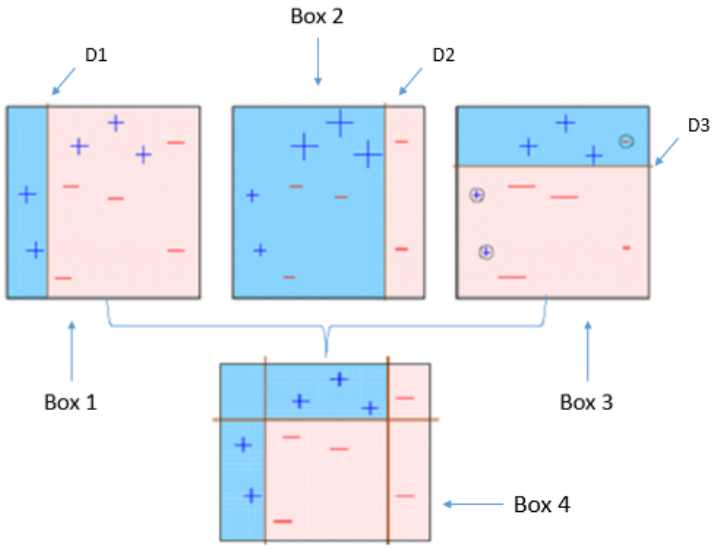

Fig. 10. Four classifiers (in 4 boxes), shown above, are trying hard to classify + and - classes as homogeneously as possible.

B. Adaptive Boosting and its variants

The algorithm below describes the most widely used form of boosting algorithm called AdaBoost, which stands for adaptive boosting.

1) Real Adaptive Boosting:

2) Gradient Boosting: Gradient Tree Boosting is a generalization of boosting to arbitrary differentiable loss functions. It can be used for both regression and classification problems. Gradient Boosting builds the model in a sequential way.

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{5}$$

At each stage the decision tree hm(x) is chosen to minimize a loss function L given the current model Fm-1(x):

$$F_m(x) = F_{m-1}(x) + \arg\min \sum_{i=1}^{n} L(y_i, F_{m-1}(x) + h(x)) \tag{6}$$

The algorithms for regression and classification differ in the type of loss function used.

C. BDTs on Market

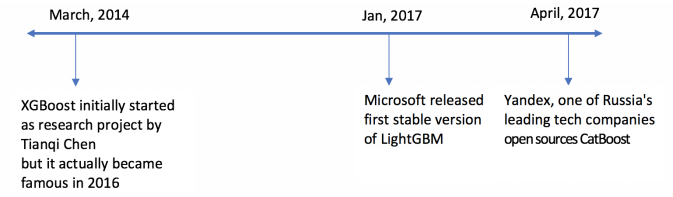

Fig. 11. Since XGBoost (often called GBM Killer) has been in the machine learning world for a longer time now with lots of articles dedicated to it, this post will focus more on CatBoost LGBM. Credit by

1) XGBoost: XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

2) LightGBM: LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages.

3) CatBoost: CatBoost is an algorithm for gradient boosting on decision trees. It is developed by Yandex researchers and engineers, and is used for search, recommendation systems, personal assistant, self-driving cars, weather prediction and many other tasks at Yandex and in other companies, including CERN, Cloudflare, Careem taxi. It is in open-source and can be used by anyone.

VI. Automated Algorithm Training

Using tree-based algorithm without parameter tuning is like driving a car without changing its gears; you can never up your speed. There are a lot of components you have to consider before solving a machine learning problem some of which includes data preparation, feature selection, feature engineering, model selection and validation, hyperparameter tuning, etc. In theory, you can find and apply a plethora of techniques for each of these components, but they all might perform differently for different datasets. The challenge is to find the best performing combination of techniques so that you can minimize the error in your predictions. This is the main reason that nowadays people are working to develop Auto-ML algorithms and platforms so that anyone, without any machine learning expertise, can build models without spending much time or effort. One such platform is available as a python library: TPOT. You can consider TPOT your Data Science Assistant. TPOT is a python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming. It will automate the most tedious part of machine learning by intelligently exploring thousands of possible pipelines to find the best one for your data.

A. Automated Machine Learning Frameworks

Automated Machine Learning provides methods and processes to make Machine Learning available for non-Machine Learning experts, to improve efficiency of Machine Learning and to accelerate research on Machine Learning. Machine learning (ML) has achieved considerable successes in recent years and an ever-growing number of disciplines rely on it. However, this success crucially relies on human machine learning experts to perform the following tasks:

TABLE III. Automated Machine Learning (AutoML) Framework Summary

Name	Language	Description
Auto-WEKA	Java	An approach for the simultaneous selection of a machine learning algorithm and its hyper-parameters. Combined with the WEKA package, it automatically yields good models for a wide variety of data sets.
Auto-sklearn	Python	An extension of Auto-WEKA using the Python library scikit-learn which is a drop-in replacement for regular scikit-learn classifiers and regressors.
TPOT	Python	A Tree-Based data-science assistant which optimizes machine learning pipelines using genetic programming.
auto_ml	Python	Automated machine learning for analytics & production. Supports manual feature type declarations.
H2O AutoML	Java/Python/R	Automated: data prep, hyper-parameter tuning, random grid search and stacked ensembles in a distributed ML platform.
Recipe	C	Machine-learning pipeline optimization through genetic programming.

B. Tree-Based Pipeline Optimization Tool (TPOT)

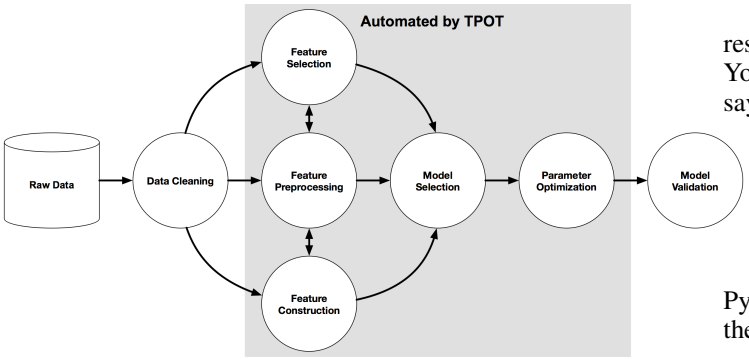

Fig. 12. Once TPOT is finished searching, it provides you with the Python code for the best pipeline it found so you can tinker with the pipeline from there.

1) Install: Use the subsubsection command with caution—you probably won't need it at, but I'm including it this an example.

	Beam Event	Cosmic	Purity	Efficiency
Adaptive	1	1	1	1
Real Adaptive	1	1	1	1
Gradient	1	1	1	1
XGBoost	1	1	1	1
LightGBM	1	1	1	1

TABLE IV. SELECTION RESULT COMPARISON

2) Selection Result Comparison:

VII. CONCLUSIONS AND RECOMMENDATIONS

Conclusion shows what knowledge comes out of the report. As you draw a conclusion, you need to explain it in terms of the preceding discussion. You are expected to repeat the most important ideas you have presented, without copying. Adding a table/chart summarizing the results of your findings might be helpful for the reader to clearly see the most optimum solution(s).

It is likely that you will briefly describe the comparative effectiveness and suitability of your proposed solutions. Your description will logically recycle language used in your assessing criteria (section **??**): "Solution A proved to be the most cost effective of the alternatives" or "Solution B, though a viable option in other contexts, was shown to lack adaptability". Do not have detailed analysis or lengthy discussions in this section, as this should have been completed in section X.

As for recommendations, you need to explain what actions the report calls for. These recommendations should be honest, logical and practical. You may suggest that one, a combination, all or none of your proposed solutions should be implemented in order to address your specific problem. You could also urge others to research the issue further, propose a plan of action or simply admit that the problem is either insoluble or has a low priority in its present state.

The recommendations should be clearly connected to the results of the report, and they should be explicitly presented. Your audience should not have to guess at what you intend to say.

APPENDIX A
PYTHON AND PYPI

Mostly of the frameworks is built on top of several existing Python libraries. So, we present how to install python 3.7 in the following:

A. Method 1: Manually Installing Python

Some users may want manually install the latest version of Python on Ubuntu by building from the source code… To do that they will need to download the installer file and run the executable.

1) Ubuntu:

- Step 1 Updating the OS:
```
sudo apt update
```
- Step 2 Installing necessary packages:
```
sudo apt install build-essential zlib1g-dev
libncurses5-dev libgdbm-dev libnss3-dev
libssl-dev libreadline-dev libffi-dev wget
```
- Step 3 Downloading the latest version:
```
cd /tmp
wget https://www.python.org/ftp/python/
     3.7.2/Python-3.7.2.tar.xz
```
- Step 4 Extracting the file and installing:
```
tar -xf Python-3.7.2.tar.xz
cd Python-3.7.2
./configure --enable-optimizations
```
- Step 5 Building process using the make command.:
```
make -j 1
sudo make altinstall
```

APPENDIX B
SET UP A JUPYTER NOTEBOOK TO RUN IPYTHON

A. Introduction

IPython is an interactive command-line interface to Python. Jupyter Notebook offers an interactive web interface to many languages, including IPython. We will walk you through setting up a server to run Jupyter Notebook as well as show you how to connect to and use the notebook. Jupyter notebooks (or simply notebooks) are documents produced by the Jupyter Notebook app which contain both computer code (e.g. Python) and rich text elements (paragraph, equations, figures, links, etc.) which aid in presenting reproducible research.

B. Installing Ipython and Jupyter Notebook

- Step 1 Installing Ipython:
  ```
  sudo apt-get -y install
  ipython ipython-notebook
  ```
- Step 2 Installing Jupyter Notebook:
  ```
  sudo -H pip install jupyter
  ```
- Step 3 Running Jupyter Notebook:
  ```
  jupyter notebook
  ```

APPENDIX C
TPOT IN PYTHON

Automated machine learning (AutoML) takes a higher-level approach to machine learning than most practitioners are used to, so we've gathered a handful of guidelines on what to expect when running AutoML software such as TPOT.

A. Installation of TPOT

The appendix is for material that readers only need to know if they are studying the report in depth. Relevcharts, big tables of data, large maps, graphs, etc. that were part of the research, but would distract the flow of the report should be given in the Appendices.

APPENDIX D
XGBOOST

Each appendix needs to be given a letter (A, B, C, etc.) and a title. LaTeXwill do the lettering automatically.

APPENDIX E
USEFUL WEBSITE

- Python Installation(Ubuntu): https://websiteforstudents.com/installing-the-latest-python-3-7-on-ubuntu-16-04-18-04/
- How to run Python script: https://realpython.com/run-python-scripts/
- Neural networks for neutrinos https://www.symmetrymagazine.org/article/neural-networks-for-neutrinos

REFERENCES

[1] D. S. Ayres *et al.* [NOvA Collaboration], "The NOvA Technical Design Report," doi:10.2172/935497

[2] P. Adamson *et al.* [NOvA Collaboration], "Search for active-sterile neutrino mixing using neutral-current interactions in NOvA," Phys. Rev. D **96**, no. 7, 072006 (2017) doi:10.1103/PhysRevD.96.072006 [arXiv:1706.04592 [hep-ex]].

[3] Tianqi Chen, Tong He "Higgs Boson Discovery with Boosted Trees" [JMLR: Workshop and Conference Proceedings 42:69-80, 2015]

[4] A. Aurisano *et al.*, JINST **11**, no. 09, P09001 (2016) doi:10.1088/1748-0221/11/09/P09001 [arXiv:1604.01444 [hep-ex]].

[5] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[6] D. Horowitz, *End of Time*. New York, NY, USA: Encounter Books, 2005. [E-book] Available: ebrary, http://site.ebrary.com/lib/sait/Doc?id=10080005. Accessed on: Oct. 8, 2008.

[7] D. Castelvecchi, "Nanoparticles Conspire with Free Radicals" *Science News*, vol.174, no. 6, p. 9, September 13, 2008. [Full Text]. Available: Proquest, http://proquest.umi.com/pqdweb?index=52&did=1557231641&SrchMode=1&sid=3&Fmt=3&VInst=PROD&VType=PQD&RQT=309&VName=PQD&TS=1229451226&clientId=533. Accessed on: Aug. 3, 2014.

[8] J. Lach, "SBFS: Steganography based file system," in *Proceedings of the 2008 1st International Conference on Information Technology, IT 2008, 19-21 May 2008, Gdansk, Poland.* Available: IEEE Xplore, http://www.ieee.org. [Accessed: 10 Sept. 2010].

[9] "A 'layman's' explanation of Ultra Narrow Band technology," Oct. 3, 2003. [Online]. Available: http://www.vmsk.org/Layman.pdf. [Accessed: Dec. 3, 2003].