JMLR: Workshop and Conference Proceedings 42:69-80, 2015 HEPML 2014

Higgs Boson Discovery with Boosted Trees

Tianqi Chen TQCHEN@CS.WASHINGTON.EDU
University of Washington

Tong He HETONGH@SFU.CA
Simon Fraser University

Editor: Glen Cowan, Cécile Germain, Isabelle Guyon, Balàzs Kégl, David Rousseau

Abstract

The discovery of the Higgs boson is remarkable for its importance in modern Physics research. The next step for physicists is to discover more about the Higgs boson from the data of the Large Hadron Collider (LHC). A fundamental and challenging task is to extract the signal of Higgs boson from background noises. The machine learning technique is one important component in solving this problem.

In this paper, we propose to solve the Higgs boson classification problem with a gradient boosting approach. Our model learns ensemble of boosted trees that makes careful tradeoff between classification error and model complexity. Physical meaningful features are further extracted to improve the classification accuracy. Our final solution obtained an *AMS* of 3.71885 on the private leaderboard, making us the top 2% in the Higgs boson challenge.

Keywords: Higgs Boson, Machine Learning, Gradient Boosting

1. Introduction

The Higgs boson is the last piece of the Standard Model of particle physics. In 2012, the ATLAS experiment (Aad et al., 2012) and the CMS experiment (Chatrchyan et al., 2012) claimed the discovery of the Higgs boson. Soon after the discovery, Peter Higgs and François Englert was acknowledged by the 2013 Nobel Prize in physics. The next step for physicists is to discover more about the physical process with the data from Large Hadron Collider (LHC). One aspect of the task is to distinguish the signal of Higgs boson from similar background processes. However, several challenges have been raised:

- The volume of data generated by LHC is huge. Every year it generates about one billion events and three petabytes of raw data.
- The signal of Higgs boson is extremely rare among background noises.
- Different physical processes are too complex to be deterministically analysed.

Machine learning is treated as one of the promising ways to tackle these challenges. To aggregate more machine learning techniques on this task, the Higgs Boson Machine Learning Challenge[1] was held. The challenge started at May 2014 and lasted for over 4 months. There were in total 1,785 teams participated in the competition, one of the largest

1. http://www.kaggle.com/c/higgs-boson

and most active ones on the platform website www.kaggle.com. The competition requires competitors to build a binary classifier for detection of signals from Higgs boson-related events.

We take a gradient boosting approach to solve these problems. Our model learns an ensemble of boosted trees which makes careful tradeoff between classification error and model complexity. The algorithm is implemented as a new software package called *XGBoost*, which offers fast training speed and good accuracy. Due to the effectiveness of the toolkit, it is adopted by many participants, including the top ones. We further improve the accuracy by introducing features that are physically meaningful. Our final solution obtained an *AMS* of 3.71885 on the private leaderboard, making us the top 2% in the Higgs boson challenge.

The rest part of the paper is structured as follows. We discuss related works in Section 2. In Section 3, we discuss the regularized boosted tree model. Section 4 focuses on details of feature engineering. Section 5 contains experimental results. Finally, we conclude the paper in Section 6.

2. Related Work

During the competition, various models are introduced. The baseline method is a naive Bayes classifier. People working with particle physics are also using tools named Multi-boost (Benbouzid et al., 2012) and TMVA (Hoecker et al., 2007). These two boosting trees implementations are used as the official benchmark scores. The model won the challenge is an ensemble of neural networks [2]. And the runner-up model [3] is based on regularized greedy forest (Johnson and Zhang, 2014).

Our approach follows the path of gradient boosting (Friedman, 2001), which performs additive optimization in functional space. Gradient boosting has been successfully used in classification (Friedman et al., 2000) and learning to rank (Burges, 2010) as well as other fields. Due to its effectiveness, there has been many software packages that implements the algorithm, including the gbm package in R (Ridgeway, 2013) and scikit-learn (Pedregosa et al., 2011). Ours differs from the traditional gradient boosting method by introducing a regularization term to penalize the complexity of the function, making the result more robust to overfitting. The advantage of regularizing boosted trees is also discussed in (Johnson and Zhang, 2014).

3. Regularized Boosted Trees

3.1. Model Formalization

In the common supervised learning scenario, the data set can be represented by a set containing n paired feature vectors and labels: $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ ($|\mathcal{D}| = n$). In the context of Higgs boson classification $\mathbf{x}_i \in \mathbb{R}^d$ is the vector of physics properties of the i-th event, while $y_i \in \{0, 1\}$ indicates whether it is a signal event.

Because the relation between the physics features and output y_i can be very complicated, a simple linear model may not capture the complicated relation between them. One possible approach is to enhance the input by dividing the input space into separate regions and model

2. https://github.com/melisgl/higgsml
3. https://github.com/TimSalimans/HiggsML

the probability distribution of y_i in each region. However, the problem remains to find such regions of interest. Ideally, we would learn them from the data. Motivated by this idea, we propose to model prediction score $\hat{y}_i$ given the input x_i by the following functional form:

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F} \tag{1}$$

where K is the number of functions and $\mathcal{F}$ is the space of functions containing the partition of the region and the score in each of them. Formally, we assume $\mathcal{F}$ to be a set of regression trees. Because our model introduces functions as parameters, it allows us to find functions $f_k \in \mathcal{F}$ that fit the data well when we train the model, thus finds corresponding regions automatically.

3.2. Training Objective

To learn the set of functions used in the model, we define the following *regularized* objective function.

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{2}$$

where l is a differentiable convex loss function that measures the difference between the prediction $\hat{y}_i$ and the target y_i. The second term Ω measures the complexity of the model (i.e., the feature functions) to avoid overfitting. One important fact about Eq. (2) is that the objective is regularized, which means we are penalizing complicated models. With the objective function, a model employing simple and predictive functions will be selected as the best model.

Because the model includes functions as parameters, we cannot directly use traditional optimization methods in Euclidean space to find the solution. Instead, we train the model additively: at each iteration t, our proposed algorithm first searches over the functional space $\mathcal{F}$ to find a new function f_t that optimizes the objective function, and then adds it to the ensemble. Formally, let $\hat{y}_i^{(t)}$ be the prediction of i-th instance at t-th iteration, we find f_t to optimize the following objective.

$$\begin{aligned}\mathcal{L}^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\mathbf{x}_i)) + \sum_{i=1}^{t} \Omega(f_i)\end{aligned}$$

This objective means that we want to add the best function to improve the model. In the general setting, the above objective is still hard to optimize. So we approximate the objective using the second order Taylor expansion.

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \sum_{i=1}^{t} \Omega(f_i)$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$. We can remove the constant terms to obtain the following approximate objective at step t.

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t) \tag{3}$$

A general gradient boosting algorithm will iteratively add functions that optimizes Eq (3) for a number of user-specified iterations. An important difference between our objective and some traditional gradient boosting method is the explicit regularization term which stops the model from overfitting. Based on this framework, we will discuss our model in details in the following subsections.

3.3. Learning the Functions

It remains to learn the function f_t in each step. In this model, we consider a specific class of functions defined by the following procedure. Firstly data is segmented into T regions based on the input $\mathbf{x}_i$, and then an independent score is assigned to each region. Formally, we define a mapping $q : \mathbb{R}^d \to \{1, 2, \cdots, T\}$ that maps the input to the index of the region, and a vector w of scores in each region. The function is defined by

$$f_t(\mathbf{x}) = w_{q(\mathbf{x})}$$

This function class includes the regression tree when q represents the decision tree structure on $\mathbf{x}_i$. We can also choose other forms of q to apply prior knowledge to regions of interest if needed.

Furthermore, we define the function complexity Ω to be the following form

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \tag{4}$$

This regularization term penalizes on the number of regions and the sum of squared scores of each region. Learning too many regions and assigning too large scores to leaves may cause overfitting thus harm the accuracy of our model. We also introduce γ and λ as two parameters to make a balance.

Define $I_j = \{i | q(\mathbf{x}_i) = j\}$ as the instance set of region j. We can rewrite Eq (3) in the following way

$$\begin{aligned} \tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \end{aligned} \tag{5}$$

Now the objective is the sum of T independent quadratic functions of elements in w. Assume the partition of regions, $q(x)$, is fixed, the optimal weight w_j^* of region j can be obtained by

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{6}$$

The corresponding optimal objective function value is

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T \tag{7}$$

We write the objective as a function of q since it depends on the structure of the mapping. Eq (7) can be used as a scoring function to score the region partition specified by q. Figure 1 demonstrates a tree with fixed structure. We learn the score w on each leaf by Eq (6).

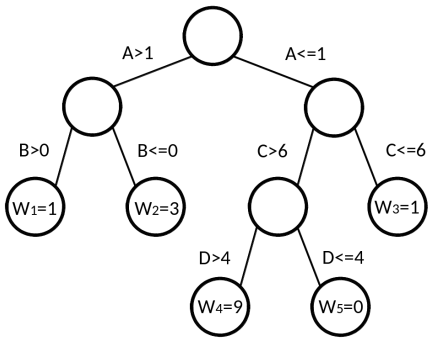

Figure 1: Demonstration of a Decision Tree

Further, we apply a generic algorithm enumerates over possible structures of q with some heuristic and output the best q we found. In the next subsection, we will discuss how to do it efficiently when q are decision trees.

3.4. Tree Growing Algorithm

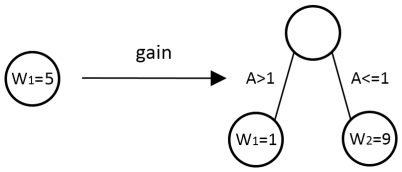

Figure 2: Tree Splitting

The objective function in Eq (7) can be used to find a good structure. Since there can be infinitely many possible candidates of the tree structure, we apply a greedy algorithm in practice. Figure 2 is an illustration of one step of the algorithm: splitting a leaf into two leaves. In each round, we greedily enumerate the features and choose to make a split on the feature that gives the maximum reduction calculated by Eq (7). More specifically, let I_L and I_R be the instance sets of left and right nodes and $I = I_L \cup I_R$, then the gain in loss

Algorithm 1: Tree Split Finding with Missing Value

Input: I, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$p
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I}, g_i$,$H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  enumerate missing value goto right
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I_k,$ *ascent order by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $gain \leftarrow \max(gain, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
  enumerate missing value goto left
  $G_R \leftarrow 0,\ H_R \leftarrow 0$
  for j *in sorted*$(I_k,$ *descent order by* $\mathbf{x}_{jk})$ **do**
    $G_R \leftarrow G_R + g_j,\ H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R,\ H_L \leftarrow H - H_R$
    $gain \leftarrow \max(gain, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split and default direction with max gain

reduction by introducing the split is calculated by

$$gain = \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma \tag{8}$$

There are two important factor that we need to consider when designing the tree growing algorithm: 1) handling missing values 2) controlling complexity. Data with missing values is a common problem in machine learning. One tradition way to handle missing values is to impute them with either mean or median of that feature dimension. However, this could require prior knowledge from the user and the specified imputed values may not be the best choice. We simply enhance the function class to learn the best way to handle missing values. The idea is to learn a default direction for each node and guide the instance with missing value along the default direction. The learning algorithm will enumerate the default directions to find the better one for each node. This algorithm is shown in Algorithm 1. It can also be viewed as learning the best imputation values, since different imputations implicitly give default directions. Algorithm 1 is recursively applied to tree nodes until some stopping condition (e.g maximum depth) is met.

Model complexity is controlled by the regularization term in Eq (4). The introduction of the complexity regularization results in the fact that the result of Eq (8) can be negative when the training loss reduction is smaller than γ. Optimizing the regularized objective

naturally corresponds to the pruning heuristic of tree structures. We use a post-pruning strategy to firstly grow the tree to maximum depth, then recursively prune all the leaf splits with negative gain. In practice, we find this results in learning more conservative trees in the latter phase of iterations and makes the result more generalizable.

3.5. Time Complexity

Recall that the number of samples is n, and the number of features is d. The total time complexity of growing one tree is $O(ndK \log n)$, where K is the maximum depth of the tree. We further optimized the algorithm by pre-sort the features at the beginning and eliminate the sorting procedure during tree growing. This will reduce the training complexity to $O(ndK)$ after preprocessing. This makes the algorithm scalable to very large dataset.

3.6. Choosing the Objective Function

Since we follow a general framework for any differentiable loss function l, we can choose an appropriate loss function for the competition task. The task is a binary classification problem, and there are also pre-assigned weights for instances in the training data. Thus we apply a weighted version of logistic loss. Assume y_i is the real label, $\hat{y}_i$ is the predicted value before logistic transformation and w_i is the weight. Then the loss function is given by

$$l(y_i, \hat{y}_i) = -w_i \left[y_i \ln \frac{1}{1+e^{-\hat{y}_i}} + (1-y_i) \ln \left(\frac{e^{-\hat{y}_i}}{1+e^{-\hat{y}_i}} \right) \right]$$

Because the data is heavily unbalanced, we find it helpful to re-balance the weight of the training data, so that sum of weight in positive and negative examples are the same. In this competition, one of the challenge is the unusual metric *approximate median significance* (AMS) (Adam-Bourdarios et al., 2014).

$$\begin{aligned} s &= \sum_{i=1}^{n} w_i I(y_i = 1) I(\hat{y}_i = 1) \\ b &= \sum_{i=1}^{n} w_i I(y_i = 0) I(\hat{y}_i = 1) \\ AMS &= \sqrt{2((s+b+b_r) ln(1 + \frac{s}{b+b_r}) - s)} \end{aligned} \tag{9}$$

where s and b are considered as unnormalized true positive and false positive rates, and $b_r = 10$ is the constant regularization term. During the competition there are attempts to optimize it with an asymptotically differentiable function (Mackey and Bryan, 2014) (Kotłowski, 2014). Since our algorithm can optimize any loss function that has the first and second order of derivations, therefore these attempts could also be combined as well. However, because AMS is an unstable measure, it could lead to the overfitting problem. So we did not choose AMS as a direct objective.

3.7. Software Package

We implemented our algorithm in C++ and make it an open source software package called *XGBoost* [4]. The package utilizes OpenMP (OpenMP Architecture Review Board, 2013) to perform automatic parallel computation on a multi-threaded CPU to give notable speedup over existing gradient boosting libraries. Besides the original C++ implementation, it also supports other languages as python, R and julia. Detail comparisons to other implementations can be found in Section 5.

Due to the accuracy and efficiency of *XGBoost*, it becomes very popular in the competition. It is used by a lot of teams to improve their performance, including some in the top 10. Thus it has an impact on the competition leaderboard. The competition administrators value the potential improvement from *XGBoost* on the current tools used in high energy physics. Therefore, *XGBoost* has won the High Energy Physics meets Machine Learning Award after the competition [5].

4. Feature Engineering

In this competition, each instance in the data refers to a simulated particle decay process after the collision. Each process is measured by a detector which gathers basic information about the decay process such as angles and momenta after the collision. To provide more information for the process, some derived features are also included. These features describe the status of particles generated in the decay process. The mission is to infer the whole image of the process from the provided description. The more we know about the status of particles, the better our inference will be.

The number of provided features in the original data is 30, which is not the complete description of the status. To have a better understanding of the process, we try to recover missing features from the given ones. We extract some additional features from the original data based on our basic knowledge of physics and the formulas mentioned in the official document. The general idea is the particles from signal events have similar kinematic status that are unique from background. Since momentum, angle and relationship between these features can describe the kinematic status well, we focus on the extraction of them with simple calculation. The list of features is as follows.

- p_x, p_y, p_z: Decomposed momentum of a particle along three axis.
- Energy of Particle: $E = \sqrt{p_x^2 + p_y^2 + p_z^2 + m^2}$.
- Vector product between any two momentum vectors i.e. p_x, p_y and p_z
  - Dot product and cosine similarity.
  - Cross product representing the plane that the two vectors span.
- Determinant product of any three vectors representing the volume that the three vectors span.

4. https://github.com/tqchen/xgboost
5. http://atlas.ch/news/2014/machine-learning-wins-the-higgs-challenge.html

- Feature extraction from sets of particles
  - Sum of p_x, p_y and p_z for each set.
  - Sum of energy for each set.
  - Sum of energy for each set, in transverse x-y plane.
  - Mass for each set.
  - Mass for each set in x-y plane.

This process generates 138 additional features. The improvement from these features is demonstrated in the next section.

5. Experiments

5.1. Setup

The data used in the competition is not from real experiments due to the extremely low number of Higgs boson signals. Instead, the data is generated from a two-stage simulation. Firstly proton-proton collisions are simulated based on the latest physics knowledge. Then the resulting particles are tracked by a virtual detector. The results from these processes maintain the statistical properties of the real events. For simplicity, there are only four types of processes retained in the simulation. Finally, 30 features are extracted from the processes. Because of different behaviors of the four physical processes, some features do not exist, this leads to some missing values indicated by -999 in the data. The final data set is split into two parts for training and test. There are 250,000 and 550,000 samples in training and test set respectively.

We next check the prediction accuracy, computation efficiency of *XGBoost* and the effectiveness of regularization. For comparison in the first two parts, we run the most popular implementations of gradient boosting machine as benchmarks: the gbm-package from R (R-gbm) and scikit-learn from python (python-sklearn).

5.2. Higgs Prediction Accuracy

In *XGBoost*, we introduce the regularization term to avoid overfitting. We also generate the default direction and take the post-pruning strategy in Algorithm 1. To exam the effectiveness, we run *XGBoost*, R-gbm and sklearn on the original training data with the same parameter setting. We set the maximum depth of the tree to 6, the step size shrinkage to 0.1 and the number of trees to 120. We also mark -999 as the missing value indicator in *XGBoost*. Since R-gbm and sklearn cannot handle the missing value, we simply keep -999 in the data.

We then fine tune the parameters of *XGBoost* to demonstrate its ability. After the training phase, we make predictions on the test data and submit to get the *AMS* score. Finally we keep the same parameter setting and run this process again on the data with additional features. The combined result is in Table 1.

We can see from the *AMS* score that *XGBoost* is more accurate under the same parameter setting. The result is greatly improved by finely tuned parameters. Comparing the two columns in Table 1, we can see the extracted physical features also have a slight improvement on the accuracy for all the models.

Table 1: Accuracy On Test Data Set

	Original Data	**Physical Features**
R-gbm	3.38356	3.38422
python-sklearn	3.55236	3.56985
XGBoost	3.64655	3.65860
XGBoost(tuned)	3.71142	3.72370

5.3. Speedup of Parallelization

In this section we exam the efficiency of of our algorithm. The comparison takes place on a CentOS 6.6 machine with Intel(R) Xeon(R) CPU E5630 and 12GB memory. We run *XGBoost* and two benchmark models on a single core with the same parameter setting as Section 5.2 on the original training data. To test the effect of parallelization, we run it on 2 and 4 cores separately with the same settings. The result is illustrated in Figure 3.

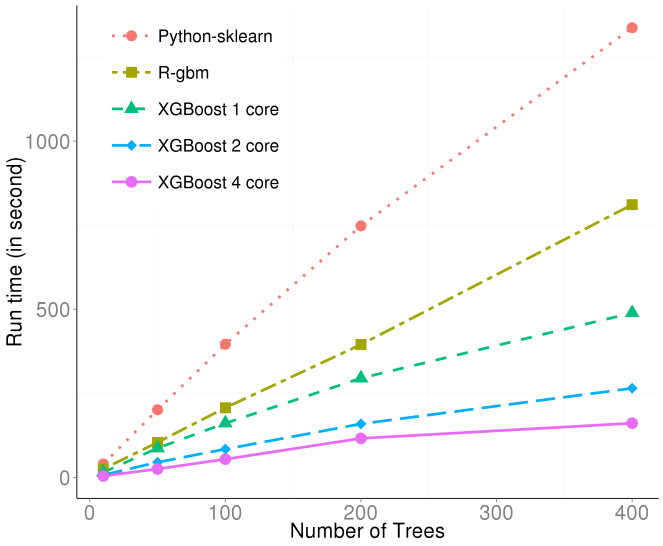

Figure 3: Speed Benchmark Result

We can see that our algorithm is faster than python-sklearn and R-gbm even only with a single thread. Additionally, our implementation also leverages multi-threading to greatly speeds up the computation. We get nearly linear speedup by using multiple threads. The result demonstrates the great advantage of our package over the existing ones.

5.4. Effectiveness of Regularization

Our finely tuned *XGBoost* result in section 5.2 sets $\gamma = 0.1$ and $\lambda = 0$. In this section we exam the effectiveness of the regularization term introduced in Eq. 4. We consider all the pairs of (γ, λ) where $\gamma = \{0, 0.1\}$ and $\lambda = \{0, 0.1, 1, 5, 10\}$. Then we train the 10 models fixing all the other parameters as the same value in section 5.2. Due to the instability of

AMS, we evaluate the performance of the models by Area Under the Curve(AUC). The result is illustrated in Figure 4

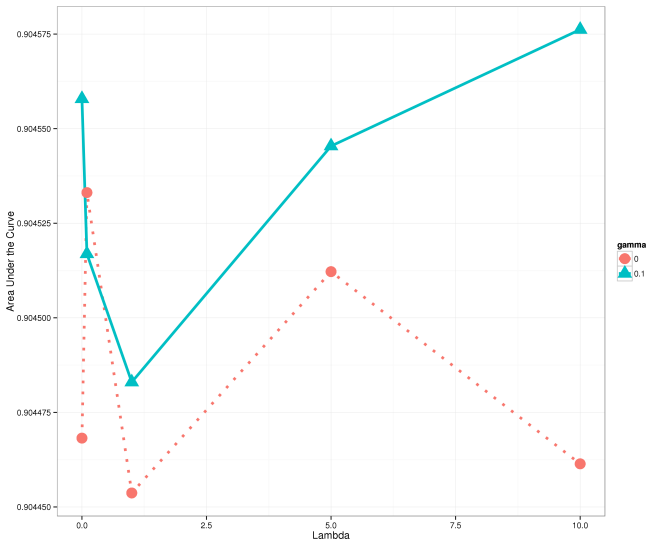

Figure 4: Regularization Comparison Result

The figure shows that nearly all models with $\gamma = 0.1$ outperformed those with $\gamma = 0$. The performance could be further improved by finer tuning other parameters with regard to γ and λ. This result demonstrates the effectiveness of the regularization.

6. Conclusion and Future Works

In this paper, we describe our solution to Higgs Machine Learning Competition. We use a regularized version of gradient boosting algorithm with a highly efficient implementation. We also take advantage of feature engineering based on physics to extract more information of the underlying physical process. Experimental results on the contest data demonstrate the accuracy and effectiveness of the techniques proposed by this paper.

One of the challenge for particle physics is the large volume of the data. To tackle this problem, the new implementation that deploys *XGBoost* to a cluster of nodes is under development. The scalability will be further improved and it will be suitable for much larger data set. It is also interesting to explore other function classes that are more physically meaningful.

Acknowledgement

We would like to thank all the competition organizers for making such an amazing challenge possible.

References

Georges Aad, T Abajyan, B Abbott, J Abdallah, S Abdel Khalek, AA Abdelalim, O Abdinov, R Aben, B Abi, M Abolins, et al. Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*, 716(1):1–29, 2012.

Claire Adam-Bourdarios, Glen Cowan, Cecile Germain, Isabelle Guyon, Balázs Kégl, and David Rousseau. Learning to discover: the Higgs boson machine learning challenge, May 2014. URL https://hal.inria.fr/hal-01104487.

Djalel Benbouzid, Róbert Busa-Fekete, Norman Casagrande, François-David Collin, and Balázs Kégl. Multiboost: a multi-purpose boosting package. *The Journal of Machine Learning Research*, 13(1):549–553, 2012.

C. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11:23–581, 2010.

Serguei Chatrchyan, Vardan Khachatryan, Albert M Sirunyan, A Tumasyan, W Adam, E Aguilo, T Bergauer, M Dragicevic, J Erö, C Fabjan, et al. Observation of a new boson at a mass of 125 gev with the cms experiment at the lhc. *Physics Letters B*, 716(1):30–61, 2012.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *The annals of statistics*, 28(2):337–407, 2000.

J.H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

Andreas Hoecker, Peter Speckmayer, Joerg Stelzer, Jan Therhaag, Eckhard von Toerne, and Helge Voss. TMVA: Toolkit for Multivariate Data Analysis. *PoS*, ACAT:040, 2007.

Rie Johnson and Tong Zhang. Learning nonlinear functions using regularized greedy forest. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(5):942–954, 2014.

Wojciech Kotłowski. Consistent optimization of ams by logistic loss minimization. *arXiv preprint arXiv:1412.2106*, 2014.

Lester Mackey and Jordan Bryan. Weighted classification cascades for optimizing discovery significance in the higgsml challenge. *arXiv preprint arXiv:1409.2655*, 2014.

OpenMP Architecture Review Board. OpenMP application program interface version 4.0, July 2013. URL http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Greg. Ridgeway. *gbm: Generalized Boosted Regression Models*, 2013. URL http://CRAN.R-project.org/package=gbm. R package version 2.1.