Detecting Higgs Boson Particles Through Machine Learning Classifiers

Yikun Zhang
Department of Statistics
University of Washington
Seattle, WA 98195
yikun@uw.edu

Abstract

In this report, we utilize two different machine learning methods to classify the Higgs bosons from background noisy particles given a Monte Carlo simulation dataset. The first predictor is the k-Nearest Neighbors (kNN) approach, which classifies a particle based on the majority vote of its k nearest neighbors. The second predictor is a powerful tree boosting method called XGBoost. By building several regression trees in a scalable way, XGBoost makes its final classification according to the additive scores of these trained regression trees.

1 Data Description and Preprocessing

High-energy particle collisions provide a modern avenue to the discoveries of exotic particles (Baldi et al., 2014). By studying the states of matter and the physical laws behind these collisions, scientists can strengthen or renew common understandings of those fundamental laws in Physics.

In this project, we build two classifiers on a subset of the original Higgs data with 100,000 instances and predict whether a particle is indeed a Higgs boson based on 27 features. Specifically, for each $i = 1, ..., n$, the first 21 attributes of a feature vector $X_i \in \mathbb{R}^{27}$ are kinematic properties measured by the particle detectors in the accelerator. The rest 6 features in $X_1, ..., X_n$ are functions of the first 21 features derived by physicists to help discriminate the Higgs bosons from background noisy particles. The responses $Y_1, ..., Y_n$ are binary, in which $Y_i = 1$ signals the Higgs boson and $Y_i = 0$ indicates the background noisy particle.

Imbalanced Data: Given the rarity of Higgs boson particles in each high-energy particle collision, we are encountering a highly imbalanced classification problem. On our training set, only 1.004% of the instances have their labels as 1. In other words, the ratio between negative and positive samples is around 98.602. In the following sections, we will discuss how to tackle such an imbalanced training set and explain our classification results by taking this factor into account.

Data Distribution: We further explore the data by visualizing the distribution of each feature in $\boldsymbol{X} = \begin{pmatrix} X_1^T \\ \vdots \\ X_n^T \end{pmatrix} \in \mathbb{R}^{n \times 27}$ through the letter-value plot; see Figure 1. The features $F_1, ..., F_{27}$ in Figure 1 are the columns of $\boldsymbol{X}$. We apply the letter-value plot (Hofmann et al., 2017) instead of the well-known boxplot to better capture the tail behavior of each feature distribution. Each indentation in a letter-value plot represents the d_k-th order statistics of the distribution for some $k \in \mathbb{N}$ defined recursively as follows:

$$d_1 = \left\lfloor \frac{n+1}{2} \right\rfloor \quad \text{and} \quad d_k = \left\lfloor \frac{1+d_{k-1}}{2} \right\rfloor \text{ for } k \geq 2.$$

As shown by Figure 1, most distributions of features are symmetric, except that $F_4, F_{22}, ..., F_{27}$ have relatively heavy right tails. The kernel density estimator (Chen, 2017) of each feature distribution is also visualized in

Preprint. Under review.

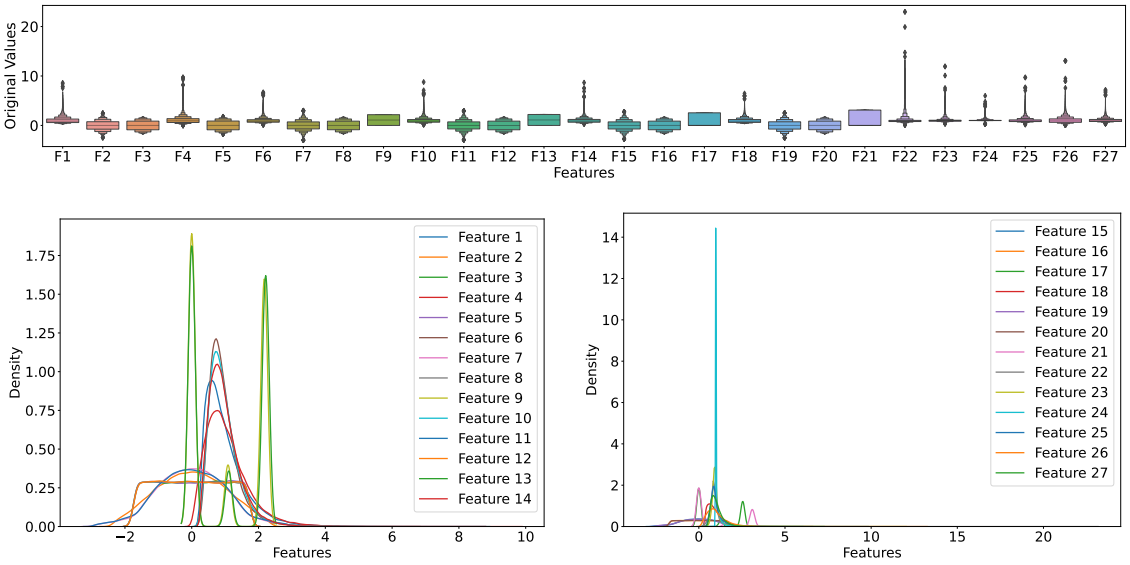


Figure 1: Letter-value and density plots of the original feature distributions.

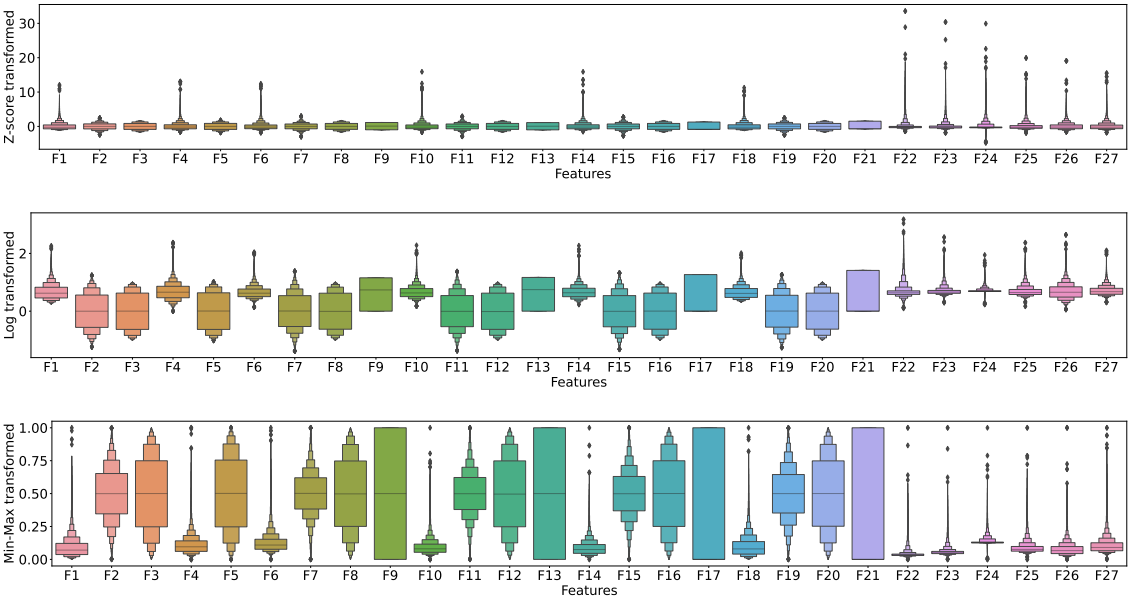


Figure 2: Letter-value plots of the z-score, logarithmic, and min-max transformations of feature distribution.

Figure 1, where it is noticeable that several features have multimodal characteristics. We consider processing the features using z-score, logarithmic, and min-max transformations:

$$F_j \leftarrow \frac{F_j - \widehat{\mu}_j}{\widehat{\sigma}_j}, \quad F_j \leftarrow \text{sign}(F_j) \log\left(|F_j| + 1\right), \quad \text{and} \quad F_j \leftarrow \frac{F_j - \min\{F_j\}}{\max\{F_j\} - \min\{F_j\}},$$

where $\widehat{\mu}_j$ is the empirical mean of feature F_j and $\widehat{\sigma}_j$ is the sample variance of feature F_j for $j = 1, ..., 27$. The transformations are applied elementwise to each coordinate of $F_1, ..., F_{27}$, respectively; see Figure 2. The patterns in feature distributions remain unchanged under the z-score transformation but become more widely spread. The log-transformation makes the feature distributions into roughly the same scale so that the outliers are closer to the main distributions. Finally, the min-max transformation pushes all the distributions to the range $[0, 1]$ and makes the distances between data points more comparable.

In Section 2, we will apply the assigned kNN predictor to the original dataset as well as the z-score, logarithmic, and min-max transformed datasets and investigate the performance differences.

2 Methodology

In this section, we describe the two predictors that are implemented to classify the Higgs boson particle.

2.1 k-Nearest Neighbors (kNN)

The kNN classifier makes its prediction by the majority vote of the k nearest neighbors of a query point (Hastie et al., 2009). To measure the distances between data points, we apply the standard Euclidean metric $d(x, y) = ||x - y||_2 = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$ in $\mathbb{R}^d$ with $d = 27$. In particular, given that there are no concrete reasons to reweigh some specific features or nearest data points, we retrieve the k nearest neighbors of a query point by the Euclidean metric and placing equal weights on all these neighboring points.

2.2 XGBoost

XGBoost (Chen and Guestrin, 2016) is a powerful gradient tree boosting algorithm that is widely used in classification tasks. It constructs an ensemble tree through K additive functions in order to make its prediction as:

$$\widehat{y}_i = \phi(x_i) = \sum_{j=1}^{K} f_j(x_i),$$

where each f_j is a regression tree. To learn the set of functions in the final model, we minimize the following objective function:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} \ell(\widehat{y}_i, Y_i) + \gamma \sum_{j=1}^{K} T_j,$$

where ℓ is a differentiable convex loss function, $\gamma \geq 0$ is the regularization parameter, and T_j is the number of leaves in each tree. Given the binary response in our classification problem, we use the log loss function:

$$\ell(Y_i, \widehat{y}_i) = -Y_i \log \widehat{y}_i - (1 - Y_i) \log (1 - \widehat{y}_i).$$

The actual training of XGBoost leverages a second-order approximation of the objective function and a greedy algorithm that starts from a single leaf and iteratively adds branches to the tree so as to learn the tree structure. To prevent overfitting, we also consider column (feature) subsampling in the training process of XGBoost models.

Due to the time constraint, it is impossible to tune every parameter of XGBoost models to the optimal. Therefore, we control the complexity of our XGBoost model by tuning the number of estimators K, the maximum depth of a tree M, the regularization parameter γ, and the feature subsampling ratio ζ. In general, the larger the number of estimators K is, the more complex the model would be. It will have a higher chance to be overfitting. In addition, increasing the maximum depth of a tree M also leads to a more complicated model. Hence, we may need a larger value of the regularization parameter γ to constrain the model complexity or subsample the columns/features when learning the tree splits. The column subsampling also helps speed up the (parallel) computations of XGBoost models (Chen and Guestrin, 2016).

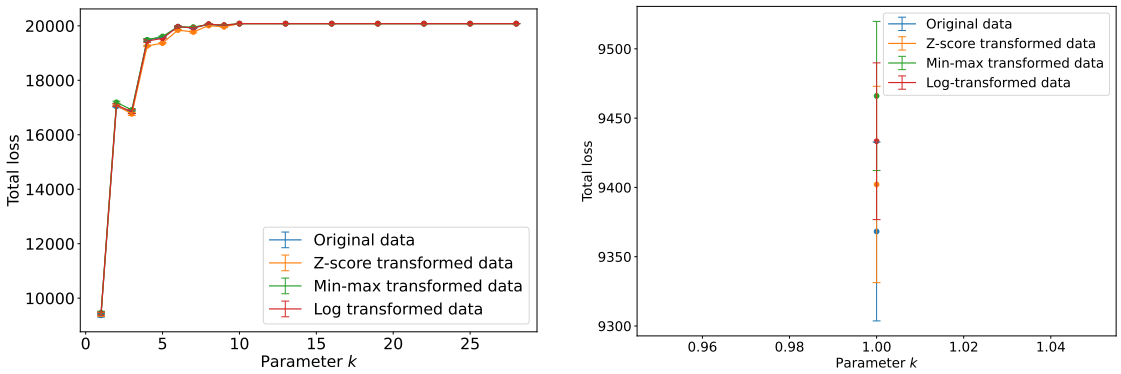
3 Training Details and Results

In this section, we delineate the details of our training processes of kNN and XGBoost predictors. The main scheme is to leverage the (5-fold) cross-validation to select the best collection of parameters for the predictor. As discussed in Section 7.12 in Hastie et al. (2009), the cross-validation error can approximate the expected error well, so it sheds light on the predictability of our fine-tuned predictor.

Evaluation Metric: Given that the responses are highly imbalanced and it is of great significance to capture every possible Higgs boson, we utilize an asymmetric classification loss to measure the performances of our predictor $\widehat{F} : \mathbb{R}^{27} \to \{0, 1\}$ as:

$$E(\widehat{F}(x_i), Y_i) = \mathbb{1}_{\{\widehat{F}(x_i) \neq Y_i, Y_i = 0\}} + 100 \cdot \mathbb{1}_{\{\widehat{F}(x_i) \neq Y_i, Y_i = 1\}}. \tag{1}$$

That is, we penalize 100 times more on a predictor $\widehat{F}$ if $\widehat{F}$ misclassifies a true Higgs boson.



(a) k ranges from 1 to 30.

(b) Zoom into $k = 1$.

Figure 3: The average total loss of kNN predictors under different choices of k and 5-fold cross-validations.

3.1 Training kNN

We randomly split our training set with 100,000 into 5 subsets with equal size, alternatively train our kNN predictor on four subsets, and measure the performances of the trained kNN predictor on the withheld subset using the asymmetric loss (1). The average total loss over all the testing data under the 5-fold cross-validation is recorded. Furthermore, to reduce the randomness of 5-fold cross-validations, we repeat the random splitting several times (due to time constraint, we only repeat 3 times) and the subsequent training and validation procedures. Figure 3a displays the 5-fold cross-validation results for kNN predictors under different choices of the parameter k, where the error bars indicate the standard errors of total losses in the repeated sampling. Notice that the total loss of the kNN predictor attains its minimum when $k = 1$. After zooming into the results when $k = 1$ in Figure 3b, we further note that the kNN predictor with $k = 1$ on the original data has the best performance (up to a certain randomness level). Therefore, we prepare the final kNN predictor by setting $k = 1$ and training it on the entire training set without using any data transformations. The total number of predicted Higgs Boson for this 1NN predictor is 99.

We evaluate the performance of this final kNN predictor through its predicted false positive rate (FPR) and false negative rate (FNR) defined as:

$$FPR = \frac{FP}{FP + TN} \quad \text{and} \quad FNR = \frac{FN}{FN + TP}, \tag{2}$$

where FP stands for the number of false positive samples, TN is the number of true negative samples, FN represents the number of false negative samples, and TP is the number of true positive samples. Again, we leverage the 5-fold cross-validation to compute these quantities and average them on these 5-fold outputs. It leads to 0.5182% for the estimated false positive rate and 45.00% for the estimated false negative rate.

3.2 Training XGBoost

Similar to the parameter tuning process of kNN predictors, we apply the 5-fold cross-validation in order to search for the best combination of four hyperparameters:

- the number of estimators K with its candidate range as $\{100, 200, 300\}$;
- the maximum depth of a tree M with its candidate range as $\{4, 6, 8, 10\}$;
- the regularization parameter γ with its candidate range as $\{0, 0.2, 0.4\}$;
- the feature subsampling ratio ζ with its candidate range as $\{0.6, 0.8, 1\}$.

In total, there are 108 possible combinations of these four hyperparameters. During the hyperparameter tuning process, we also explore the case when $K = 400, 500$ because the resulting XGBoost models under these choices of parameters overfit the data. The implementation of XGBoost in `Python` also provides a hyperparameter for balancing the positive and negative classes. We set it 98, which is the ratio between negative and positive samples. By specifying this hyperparameter, it scales the gradient for the positive class

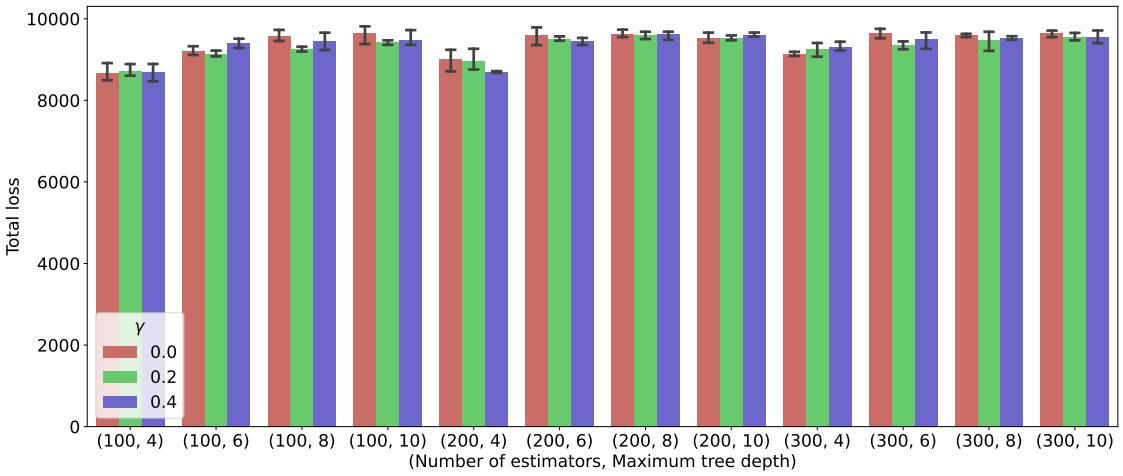


Figure 4: The average total loss of XGBoost predictors under different combinations of the hyperparameters and 5-fold cross-validations.

and helps the model achieve better performance when making predictions on the positive class[1]. (Rigorously speaking, as we use (1) as the evaluation metric, we should have set this hyperparameter to be 100. We use the default as the ratio between negative and positive samples in that the value 98 is close to 100 and the performances of resulting XGBoost models are similar.)

Among the 108 possible combinations of the above four hyperparameters, the parameter set $(K = 100, M = 4, \gamma = 0.4, \zeta = 0.8)$ attains the minimal average total loss under 5-fold cross-validations; see Figure 4. In Figure 4, the error bar in each bar plot is the standard error among different choices of the feature subsampling ratio ζ, so the one with the smallest lower error bound will indicate the optimal value. We estimate the predicted false positive rate of the final XGBoost model under the optimal choices of hyperparameters as 6.08% and predicted false negative rate as 34.701% through a 5-fold cross-validation.

4 Conclusion

In this report, we explore the performances of two different predictors on the Higgs boson data. Different from the original paper (Baldi et al., 2014), which utilizes a 5-layer neural network model and trains it on a roughly balanced data sample, our task is more challenging due to the imbalanced labeling issues and time constraints. It is worth mentioning that the 1NN classifier works well on this imbalanced labeling data under a rigorous 5-fold cross validation procedure. Finally, by searching over numerous combinations of the hyperparameters, the powerful XBGoost model can further reduce the predicted false negative rate by more than 22% when compared with the optimal 1NN predictor, though it comes with a sacrifice in wrongly predicting many Higgs boson.

References

P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5(1):1–9, 2014.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

Y.-C. Chen. A tutorial on kernel density estimation and recent advances. *Biostatistics & Epidemiology*, 1(1): 161–187, 2017.

[1]See https://machinelearningmastery.com/xgboost-for-imbalanced-classification/ for reference.

T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

H. Hofmann, H. Wickham, and K. Kafadar. Letter-value plots: Boxplots for large data. *Journal of Computational and Graphical Statistics*, 26(3):469–477, 2017. URL https://doi.org/10.1080/10618600.2017.1305277.