Signal-Verses-Background Classification of Higgs Boson Using Machine Learning Methods

Yiding Qiu

University of California San Diego

yiqiu@ucsd.edu

Abstract

The study of experimental particle physics requires distinguishing exotic particles generated by particle collision with the background noises. The non-linear function in hyperspace gives major obstacles in classification when using data analysis and machine learning methods. In this paper, several different classification methods are used on the benchmark datasets of Higgs bosons and supersymmetry particles and are evaluated based on their performance on the dataset. The result shows significant improvement in accuracy when using deep learning neural network.

1. Introduction

In high energy particle physics, specifically focus on exotic particles finding, machine learning methods are used in signal-background discrimination. The sensitivity of the testing, or false negative rate, is essential in searching for the best method. The main purpose of the experiment is to discover new particles, which has a huge economical cost and at same time is time consuming, and therefore missing the finding is most undesirable. For early researches of this field, the most common methods are boosted decision trees, matrix elements and feed-forward neural networks with one to two hidden layers.[1] With the development of machine learning in recent years, using deep learning neural network with multiple layers is claimed to improve the classification metric by as much as 8% (Baldi, Sadowski and Whiteson ,2014)[2] With the aid of data analysis performed by machines, human skills in deriving specific features from observed date are less required, which also simplifies tasks.[3]

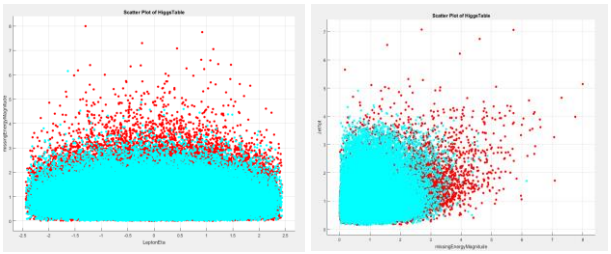
Different methods of data analysis are used to test the hypothesis-- that using deep learning neural network, the accuracy will be increased. Logistic regression is tested using Matlab as a baseline. Other methods are tested with Matlab using pre-installed toolbox, including boosted tree and neural pattern recognition. The deep learning method is performed using python and Theano, along with Pylearn2 package. The original code of deep learning neural networks performed on benchmark dataset is credit to Peter Sadowski. In this paper, I propose that using deep learning, the accuracy will be increased by at

least 5 percent, which is significant regarding the difficultness in distinguishing the background signal with the actual signal; In addition, other methods retains similar accuracy, and is most likely due to the complexity of datasets. The result presents consistency with the result from the article "Searching for Exotic Particles in High-Energy Physics with Deep Learning", which confirms the improved accuracy using deep learning for these datasets.

2. Data Description

The dataset is downloaded from UCI Machine Learning Website. The HIGGS dataset is produced by Monte Carlo simulation, a technique simulates probability and used to understand the impact of risk and uncertainty. The dataset contains 28 features in total. First 21 features are kinematic properties measured by the particle detectors in the accelerator. The last seven features are high-level features derived by physicists, which are functions of the first 21 features that help to discriminate between two classes. The original data contains 11000000 data points. Here only the first 150000 data points are used due to the limited memory of the computer. All features are normalized. The outcome is binary and assigned to 0 for background and 1 for signal.

One straightforward way to see whether some features standout in classifying background noise and signal can be performed using Matlab (Classification Learner). The HIGGS dataset demonstrates a complex overlap of features, and therefore it is impossible to find some less relevant features. Graph 1and 2 are scatter plots of one feature versus the other; the red dot indicates the background (0) and blue dots indicates signals (1). These graphs shows the interconnections of the features and most of them has overlap part for either backgrounds or signal. Combination of different features are examined and no evidence shows any characteristic features.



3. Method

Logistic Regression, boosted decision tree, neuro pattern recognition are performed on the dataset using Matlab. Shallow neural network with one hidden layer and deep neural network with four layers are tested using Theano. Using each method, three types of test are performed. One uses only low-level features, one uses high level features, and one uses both low and high levels of features. The accuracy is calculated based on the internal algorithm of Matlab functions. Area under the curve (AUC) is also calculated using built-in Matlab function and Theano function depending on different platforms that used. AUC is largely used in ranking different machine learning methods. It measures the skill of classifiers depending on to the specific threshold. The curve here refers to ROC curve, and it is plotted using the true positive rate verses false positive rate. Generally speaking, larger the value of AUC, better the classifier is ranked.

3.1 Logistic Regression

Since the training is supervised, and the output is binary, logistic regression is performed by Matlab built-in function (mnrfit and mnrval). The dataset is randomly sampled to 70% of training data and 30% testing data. A matrix of regression coefficients are generated using the training data, and can be

plugged in to the other function to calculate the probability of each class

in the testing data. Using the for-loop, the probability greater than 0.5 is assigned to signal (1) and else are grouped as background (0). The accuracy is simply calculated as comparing the predicted value to the actual outcome, and count the same output. In Matlab, the logistic regression if linear. The equation is as following.[4]

$$\operatorname{logit}(\mathbb{E}[Y_i \mid x_{1,i},\ldots,x_{m,i}]) = \operatorname{logit}(p_i) = \ln\left(\frac{p_i}{1-p_i}\right) = \beta_0+\beta_1 x_{1,i}+\cdots+\beta_m x_{m,i}$$

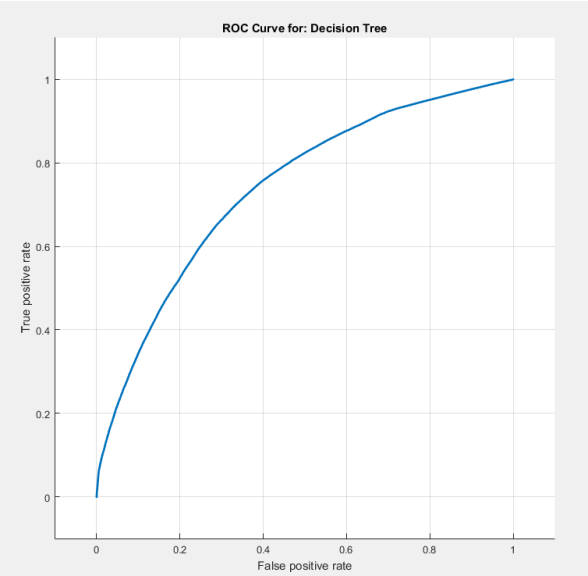
The accuracy and area under the curve is calculated using all features, only low features and only high features

	Accuracy	AUC
All features	64.3%	0.683
Low-level features	55.5%	0.592
High-level features	61.8%	0.648

3.2 Decision Tree

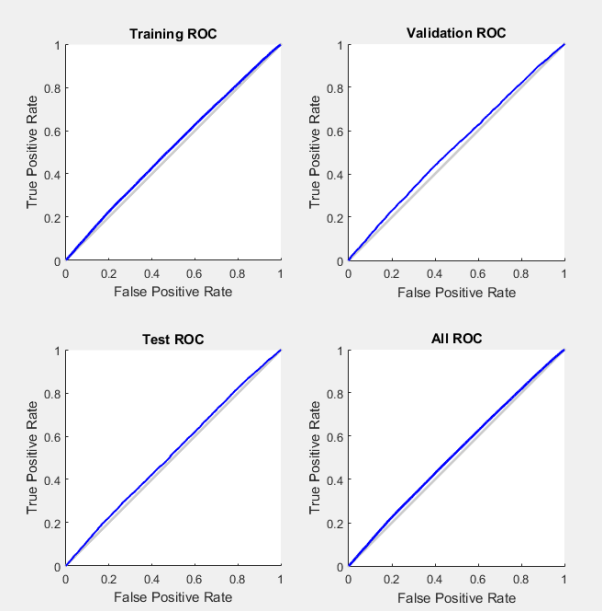
Using the toolbox in Matlab, Decision Tree can be used to classify the dataset. 5 folds of cross validation is performed by the function automatically. This method uses nodes to show tests on an attribute. Each branch represents the outcome of the test and each leaf node represents a class label. It can be presented as a flowchart-like structure. The paths that from root to leaf represents classification rules. The accuracy and area under the curve is calculated using all features, only low features and only high features.5

	Accuracy	AUC
All features	68.2%	0.741
Low-level features	60.3%	0.632
High-level features	67.4%	0.734



3.3 Neural pattern recognition using Matlab

This method is used to check how the built-in function in Matlab using neural pattern recognition performed with the dataset. Simply gives the data of input and target, the training performed with on hidden layer. Multiple values of hidden neuron numbers are tried, with only small differences. Using 70% training set, 15% validation set and 15% test set, the accuracy is 62.9% (100%- error rate indicated by the function) The ROC curve is close to the orthogonal baseline, which mean the AUC is close to 50%.



3.4 Shallow and Deep Neural Network

The basic idea of using neural network is to use hidden layers that enables computer to learn from data. It is possible for neural network to simulate any functions. However, the main shortcut is that it is prone to overfitting problems, and is also time consuming comparing to other simpler algorithms.

Python is used to analysis the data. The code is obtained on the internet,6 which is provided by the authors of the paper "Searching for Exotic Particles in High-energy Physics with Deep Learning." Nature Communications 5 (July 2, 2014). Since a smaller dataset is extracted from the original dataset, code is changed accordingly. In addition, some minor errors are presented in the original code, and was modified in order to run correctly.

The python code uses Pylearn2 and Theano to train the neural network. Since Pylearn2 provides models of training, it only requires to give arguments and set initial values. The whole dataset is divided to training (70%), testing (15%), and validation (15%) sets. For both shallow and deep neural network models, the training is set with 42 seeds and 1000 kernel width. The initial momentum is 0.9 and increases to the maxmum of 0.99. The training process stops until the minimum error of the validation set decrease 10^-5 over 10 epochs. Usually training uses 400-500 epochs. Error rate and AUC is also calculated using built-in function in Theano.

Shallow neural network	**Accuracy**	**AUC**
All features	65.7%	0.754
Low-level features	52.8%	0.658
High-level features	62.3%	0.704

Deep neural network	Accuracy	AUC
All features	70.2%	0.802
Low-level features	62.7%	0.782
High-level features	62.5%	0.734

Results

Comparing the overall accuracy using all features and area under the curve (Table 1), it is clear to see that using deep neural network with 4 layers performs the best, and using neural network with one layer also has better performance. Logistic regression has the worst performance and can be a benchmark for tests using other methods. Surprisingly, the Boosted Decision Tree also gives high accuracy and fairly large AUC. Also, consistent with the other paper using the same method, low-level features has larger AUC than high-level features when deep learning network is used. This suggests

that using deep learning method, it is not required for humans to derive the non-linear function manually.

	Accuracy	AUC
Logistic Regression	64.3%	0.683
Boosted Decision Tree	68.2%	0.741
Neural pattern recognition	62.9%	~ 0.53
Shallow Neural Network	65.7%	0.754
Deep Neural network	70.2%	0.802

Conclusion

In this paper, data analysis using different methods are performed, and the hypothesis is proved. Deep learning neural network is useful in background-signal discrimination, and is especially useful when low-level features are used. This does not only lessen the data needed, but also alleviates work-load for physicists.

Besides accuracy, calculation time and cost is also essential in evaluation the classifier. The time is not accurately measured, but the differences between those methods are significantly huge. The logistic regression and neural pattern recognition only took less than one minute; the Boosted Decision Tree uses 2 or 3 minutes for training; shallow Neural Network and Deep Neural network took several hours to complete training. One major limitation here is the overall performance of computer. All computers and laptops used for the experiment rely on CPU, which significantly slow the calculation process. Also the memory of computers are limited, and therefore only a small dataset (around 100 MB) is used.

References

[1] Thomas R. Junk. The University of DØ lecture series http://www-cdf.fnal.gov/~trj/ud02012/ud02012trjstats_day2.pdf

[2] Baldi, P., P. Sadowski, and D. Whiteson. “Searching for Exotic Particles in High-energy Physics with Deep Learning.” Nature Communications 5 (July 2, 2014).

[3]University of California - Irvine. "'Deep learning' makes search for exotic particles easier: New computing techniques could aid hunt for Higgs bosons." ScienceDaily. ScienceDaily, 2 July 2014.

<www.sciencedaily.com/releases/2014/07/140702093606.htm>.

[4] https://en.wikipedia.org/wiki/Logistic_regression

[5]Cha, S., & Tappert, C. (n.d.). A Genetic Algorithm for Constructing Compact Binary Decision Trees. JPRR Journal of Pattern Recognition Research, 1-13.

[6] https://github.com/uci-igb/higgs-susy

Appendix

Argue for bonus points

The dataset is pretty large regarding to the capacity of Matlab and my laptop. Also I spend many hours trying to fix the minor problems of the code provided. Also I spend a lot of time trying to test Pylearn2 and made it work. I think I learned a lot trying to understand the code for deep learning neural networks. The dataset I used is pretty interesting too.

Matlab Code:

```
function [accuracy,AUC ] = LogisticReg(lable,feature)
%calculate accuracy,error,and AUC of the dataset
%   dataset is radomly sampled with 70% of train data and 30% testing data
dataset = cat(2,lable,feature);
train = datasample(dataset,0.7*size(dataset,1),1);
train_lable = train(:,1)+1;
[B,dev,stats] = mnrfit(train(:,2:end),train_lable);

test = datasample(dataset,0.3*size(dataset,1),1);
pihat = mnrval(B,test(:,2:end));

for i = 1:size(pihat, 1)
    if pihat(i,1)> 0.5
    Pred1(i,1) = 0;
    else
    Pred1(i,1) = 1;
    end
end

accuracy = sum(Pred1 == test(:,1))/size(test,1);
error = sum(lable==1)/size(lable,1)*accuracy + sum(lable==0)/size(lable,1)*(1-
accuracy);

mdl = fitglm(feature,lable,'Distribution','binomial','Link','logit');
scores = mdl.Fitted.Probability;
[X,Y,T,AUC] = perfcurve(lable,scores,1);

end

[ac_tol,AUC_tol] = LogisticReg(HIGGS_y,HIGGS_x);
[ac_low,AUC_low] = LogisticReg(HIGGS_y,HIGGS_x(:,1:21));
[ac_high,AUC_high] = LogisticReg(HIGGS_y,HIGGS_x(:,22:28));
function [trainedClassifier, validationAccuracy] = trainClassifier(datasetTable)
% Extract predictors and response
predictorNames = {'VarName2', 'VarName3', 'VarName4', 'VarName5', 'VarName6',
'VarName7', 'VarName8', 'VarName9', 'VarName10', 'VarName11', 'VarName12',
'VarName13', 'VarName14', 'VarName15', 'VarName16', 'VarName17', 'VarName18',
'VarName19', 'VarName20', 'VarName21', 'VarName22', 'VarName23', 'VarName24',
'VarName25', 'VarName26', 'VarName27', 'VarName28', 'VarName29'};
predictors = datasetTable(:,predictorNames);
```

```
predictors = table2array(varfun(@double, predictors));
response = datasetTable.VarName1;
% Train a classifier
trainedClassifier = fitctree(predictors, response, 'PredictorNames', {'VarName2'
'VarName3' 'VarName4' 'VarName5' 'VarName6' 'VarName7' 'VarName8' 'VarName9'
'VarName10' 'VarName11' 'VarName12' 'VarName13' 'VarName14' 'VarName15' 'VarName16'
'VarName17' 'VarName18' 'VarName19' 'VarName20' 'VarName21' 'VarName22' 'VarName23'
'VarName24' 'VarName25' 'VarName26' 'VarName27' 'VarName28' 'VarName29'},
'ResponseName', 'VarName1', 'ClassNames', [0 1], 'SplitCriterion', 'gdi',
'MaxNumSplits', 100, 'Surrogate', 'off');

% Perform cross-validation
partitionedModel = crossval(trainedClassifier, 'KFold', 5);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');

%% Uncomment this section to compute validation predictions and scores:
% % Compute validation predictions and scores
% [validationPredictions, validationScores] = kfoldPredict(partitionedModel);
```

Python code is similar to the one in the github, only with some modifications