Getting over 84% in finding the Higgs boson

Baetu Ciprian, Volodymyr Lyublinets, Musuroi Doru

***Abstract*—Facing the challenge to detect the appearance of a Higgs boson in a collision of protons, we perform an analysis of the provided features, then we propose a featurization pipeline on the result of which we employ different models as baseline evaluation and a bag of neural networks model that performs the best.**

I. Introduction

Throughout the month of the competition we've made gradual progress towards reaching a satisfying result. We start with no featurization and basic models as linear regression, ridge regression and logistic regression. These models prove to have a rather small accuracy on a cross-validation evaluation, Table I. We then turn our heads towards feature augmentation and add a polynomial basis of degree 2 for each feature. As expected, the accuracy increases as presented in Table I. Then we dive deeper into data analysis and we try to understand the features and how to combine them. This results in the final feature augmentation pipeline presented later in the report. The results can be seen in the Table I.

While the above models are good for certain basic tasks, they are rarely used for more complex datasets such as this one, where the relationship between the inputs and outputs is not trivial. Neural networks have shown to have good performance on complex datasets due to their ability to bend the decision boundaries - something that basic regressions can only achieve with advanced featurization. Since a basic version of a neural network takes slightly more than 100 LoC to implement, it's smarter to use it rather than spend days trying to pick good features.

For the above models have to fight issues like overfitting, underfitting or large training times, that prevents us from effectively using the chosen models. We prevent overfitting by use of regularization. We tackle the large training times of neural networks by changing the learning method from gradient descent to stochastic gradient descent which proved to be a great improvement.

II. Data analysis

Before employing any model, we proceed to do data imputation and cleaning procedures.

To start with, the provided dataset has a problem with missing values. After basic exploratory analysis, we see that missing values are split into three groups and either the entire group is missing or the entire group is present. These column groups are:

First group
- DER_[deltaeta,mass,prodeta]_jet_jet,
- DER_lep_eta_centrality,
- PRI_jet_subleading_[pt,eta,phi],

Second group
- PRI_jet_leading_[pt,eta,phi],

Column 0
- DER_mass_MMC

After reading the documentation **ADD HERE LINK TO BIBLIOGRAPHY**[2], we find that these values are not missing at random, but rather undefined when PRI_jet_num ¡= 1 for the first group of columns, undefined when PRI_jet_num = 0 for the second group of columns and undefined when the event is too far from the expected topology for DER_mass_MMC. Column 0 (DER_mass_MMC) is particularly interesting - if we look at b/s ratio for cases when it is undefined, we see that over 90% have the 'b' value. Thus, this is a very important signal in itself and we should not just discard it.

We have tried various strategies for what to put in place of missing values - means, medians, max value + ϵ and even some class-based replacement (all over columns). Out of all these, using means has shown the best performance for neural networks. It is worth mentioning that NNs are less finicky to replacement strategy than other models - for example they correctly recognize that most column 0 entries with missing values should be a 'b' as long as we impute the same value for training and testing datasets.

Lastly, before proceeding to featurization, we have to standardize the data. This is crucial for models like linear regression, where without this low-magnitude columns will just get ignored. This is also important for neural networks, where having data in Gaussian(0, 1) form improves performance too. It's worth mentioning that we compute means and variances for standardization across merged training and test datasets (without missing values) and use the those to standardize both. Otherwise, we could suffer for distribution differences between some columns in train and test, leading to poor results.

III. Feature augmentation

Throughout the competition we've tried many variants of the new features and chose the ones that have shown to provide the biggest benefits to our submission model. In the beginning, the process of deciding whether a new set of features was good involved doing a simple cross validation,

but when result crossed the 0.84 mark, we started using 5-fold cross-validation to battle variance. The final list of used features is the following:

Squares of the original features values
: Adding them to linear regression shows an improvement from 0.74 to 0.77. This is quite expected, as adding polynomial features increases the representational power of linear models. We then try adding them to a 1 hidden layer NN and the result goes up from X to Y. Neural network never multiplies the data with itself, so it's reasonable that squared features are useful for it too. Adding higher power features does not lead to additional improvements for our neural network.

Cosines of angles and pairwise angle differences

Converting PRI_jet_num column using one-hot encoding
: This is the only categorical feature in the dataset with only 4 options.

Radial Basis Features
: For columns X and Y that are G(0, 1), we add column with $exp(-||X - Y||^2/2)$. RBF features are typically used to extend Support Vector Machines. We tried adding them to our neural network, which resulted in a minor improvement.

As a note to the rather concise feature augmentation pipeline, using a neural network allows us to pus less effort into this process compared to using a linear model. In this reason, neural networks are known to have ability to fit complex data shapes as a result of non-linearity, thus sparing us from searching for features that can allow simple regressions to achieve this behavior.

IV. Benchmarks

In our gradual progress, we hit a few milestones that are defined by different results in the evaluation of our models. We present in Table I hereunder the accuracies each model used by us achieved. For the linear and ridge regression we used the normal equations to get the minimum weight corresponding to the mean squared loss. For the latter one, we used regularization parameter $\lambda = 0.01$. For logistic regression, we used full gradient descent in order to optimize the maximum likelyhood criterion. Over all the runs, we used a regularization parameter $\lambda = 0.01$, weights initialized to zero, 500 iterations and a learning step $\gamma = 1e^{-6}$

Linear regression	67.064%	68.380%	79.301%
Ridge regressions	67.062%	68.382%	79.305%
Regularized logistic regression	74.881%	77.369%	79.067%

Table I
RESULTS OBTAINED ON A 3-FOLD CROSS-VALIDATION EVALUATION FOR THE SPECIFIED MODELS USING DIFFERENT APPROACES FOR FEATURE AUGMENTATION

V. Final model

Our final model is based on neural networks and we would like to highlight the most interesting details behind its training and architecture:

- The network is composed of interchanging fully-connected and ReLU layers (with no ReLU at the very end). We allow arbitrary number of hidden layers. ReLU was chosen as an activation function due to its simplicity - backpropagation is very easy to implement for it.
- We use the softmax loss function - we've tried both hinge and softmax losses, with a tiny margin softmax performed better.
- Originally, we were using full gradient descent since the data was small and fit into RAM. However, we later discovered that with minibatches we can train the network to achieve the same validation score in 5x less time! Additionally, we use RMSprop update rule for the weights - this helped to speed up convergence dramatically as well.
- We found that 2+ layer networks achieve roghly the same performance in the optimal condition. Thus, we chose to stick with 2 layer neural networks as they are faster to train.
- At one moment we found a very interesting problem with NNs - we were unable to train NNs with 2+ hidden layers of certain sizes, especially those were hidden layer size was small (under 50 "neurons"). With small initial weights the network just wouldn't get trained (accuracy after 100 iterations is the same as in the beginning) and with large weights the error would go to infinity. The reason for this lies in poor weight initialization - in the former case at the last layer we end up with essentially zeroes, while in the latter the numbers are enormous. Ideally, one uses a batch-normalization layer to avoid this, but it is non-trivial to implement. Thus, we dealt with this problem by good weight initialization, where instead of just making each weight be Gaussian(0, 1) times a constant, we additionally divided it by the square root of the number of layer inputs (ADD CITATION). See the reference for detailed explanation of why this helps, it's quite interesting.
- We optimized each of the NNs parameters using grid search - since this could take hours in some cases, we used a remote AWS server.
- The best NN that we trained achieved around 84.4% +/- 0.2%. To further reduce variance, our final submission is a bag of ??? such neural networks. Each of these NNs was trained on 80% of the data. Afterwards,

the predictions of these networks are combined using majority voting.

VI. Conclusion

Throughout the course of the competition we not only applied algorithms seen in class to a real-world dataset, but also learned about other models and techniques, such as decision trees, neural networks and bagging. We discovered many practical details of using neural networks. While we stopped short of our goal of building ensembles of decision trees and neural networks, we still ended up with a model that allowed us to hold the #1 spot during the entire competition.