Higgs Boson Machine Learning Challenge - technical report

Pawel Golinski, Igor Katselenbogen, Josef Weber
EPFL, Switzerland

***Abstract*—We present our machine learning system and approach used for solving *Higgs Boson Machine Learning Challenge*. Several methods for training predictive model are evaluated. To improve accuracy, data is preprocessed accordingly before inputting it into algorithm of choice. We run several experiments to pick hyperparameters. Final accuracy achieved is 0.804.**

I. Introduction

In 2014 kaggle launched *Higgs Boson Machine Learning Challenge*. The competition explored possibility of using machine learning methods in particle physics research. As part of the contest, the dataset containing descriptions of thousands of particle collision events was published. Each event was labelled either as signal or as background, depending on whether the event may contain data leading to discovery of new particles (signal) or not (background). The goal of the challenge was to label the unlabeled events.

This report describes our approach used for solving the challenge, adapted for needs of EPFL's *Machine Learning* course project. First, we briefly explore dataset. Then we describe all machine learning algorithms considered for solving the problem and explain our reasoning behind our algorithm of choice: logistic regression. We also provide high level descriptions of our machine learning system. Finally, we describe experiments that led us to appropriate hyperparameter choice. Our final kaggle score is 0.804.

II. Dataset

The dataset contains 30 learnable features and 250000 training examples. The examples are labeled by 's', if they are positive example, and 'b' if they are negative example. The problem is therefore a straightforward classification problem - we want to predict a label for each example based on all 30 features available.

Most of the features are of floating point type, although since they are different physical measurements, the units and semantics may vary. Some feature engineering was already done before publishing dataset - it contains features prefixed with *PRI* and *DER*, corresponding to primitive and derived features. Derived features represent certain physical properties inferred from primitive features [1].

There are a lot of missing values - if the feature is undefined, its value is set to -999.0. We decided to set these undefined values to zeros. It gave a slight boost in our accuracy when compared to other methods for dealing with missing values, e.g. inputting median or mean of all non-missing values. It intuitively makes sense - 0s cause model parameter for corresponding feature to be ignored. This is similar to known regularization technique - dropout [2].

The only non-float learning feature is *PRI_jet_num*. It is a categorical feature with integer value in range 0-3. Some other features' semantics is determined by value of this variable. There are several ways of dealing with categorical variables. We tried changing them to four binary 0/1 valued features, but it didn't give any improvement to accuracy. However, after deciding to train 4 separate models, each one for separate *PRI_jet_num* value, we observed accuracy boost of 2-3%.

III. Chosen approach

There were several methods we considered using for the Higgs Boson challenge: four different versions of linear regression and two versions of logistic regression. Intuitively, it makes sense to choose logistic regression as a natural method for solving classification problem, but we also evaluated other methods to make sure it is the right choice.

Our approach for this problem was as follows - we first tried all algorithms on initially preprocessed dataset (with unknown -999.0 placeholders changed to zeros). Then, before proceeding with more complicated techniques for data preprocessing, which would be training method specific, and starting hyperparameter tuning, we decided to choose as a baseline the model that would reach the best accuracy in this early stage.

Before each evaluation, the dataset was split randomly into train and validation dataset. Models were only trained on train dataset and then evaluated on both training and validation dataset in order to allow us to detect overfitting and use regularization if needed.

The methods can be divided into two categories: one-shot methods that give you the answer right away and the options for tuning them are limited and iterative methods, which need more time to converge, but allow us to examine convergence process and make further decisions on dataset preprocessing or hyperparameter tuning based on it.

One-shot methods we tried were least squares and ridge regression. Out of these two methods, ridge regression gave us better accuracy of 0.74458, which was our initial kaggle submission, and our "baseline" score to beat. After further experiments we decided to abandon these methods - iterative techniques found better optima and we also sometimes observed a strange effect of final test score on kaggle being much worse than validation score, a symptom of overfitting.

Main iterative approaches we considered were linear and logistic regression, with several variants based on optimization and regularization techniques. To develop baseline using iterative techniques we ignored the problem of possible overfitting, to be addressed later, focusing instead of making sure we choose method with the best convergence on training dataset.

There are several optimization techniques that can be used with iterative methods - we examined Stochastic Gradient Descent and Gradient Descent. We noticed Gradient Descent was much faster to converge for both linear and logistic regression. SGD was susceptible to outliers, which sometimes caused loss to temporarily increase. Using GD guarantees convergence with appropriate *gamma* parameter, since it uses entire dataset before updating parameters. That is why we chose GD as our optimizer.

Out of GD implementations of linear regression and logistic regression we were able to get better result with logistic regression and the correlation between accuracy and loss was much better. It is natural - logistic regression predicts probability value in range [0, 1], and linear regression predicts value from range [-inf, +inf], which sometimes causes loss to become bigger than necessary.

IV. Machine learning system

Our machine learning system contains three Python scripts: *run.py*, *dataset.py*, *implementations.py*. Following good practices of software development, each of these scripts has different, specialized objective.

The first script, *run.py*, is our executable script which generates kaggle prediction. It uses functionality of other two scripts to load and prepare dataset, launch training, evaluate it, display final loss and accuracy and generate test predictions.

The other scripts are "library" scripts containing functionality for operating on dataset and training models and using them for evaluation. *dataset.py* contains dataset loading, preprocessing and train/validation dataset splitting code. *implementations.py* contains implementations of machine learning methods. It also implements Training object, which wraps all methods under one common API, inspired by scikit-learn library. This allows for fast trial-and-error iteration, since the method used and its hyperparameters are defined in one region of code and can be easily changed to launch additional experiments.

V. Experiments

In section III, we described our reasoning behind choosing logistic regression as our algorithm of choice. Now, we describe our method of finding optimal hyperparameters and data preprocessing scheme to improve baseline score.

An initial implementation of logistic regression gave us accuracy of around 0.73. A common normalization technique usually used with logistic regression is mean centering and standard deviation division, this caused us to reach accuracy of 0.75038, which become our baseline to beat.

A technique that gave us the biggest accuracy boost was polynomial feature expansion - we used degree of 8, since we

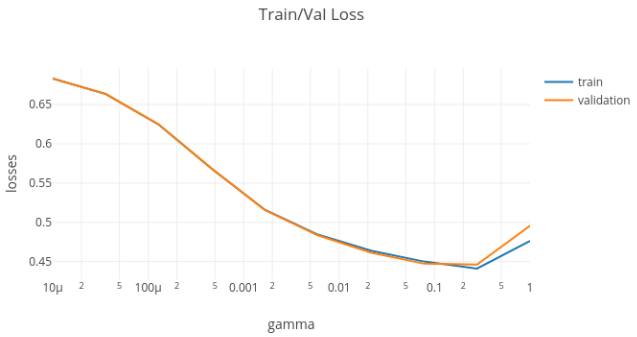

Fig. 1: Comparison of training and validation loss, with varying *gamma* and *n_iters* = 3000.

didn't observe greater improvements with bigger degrees and they sometimes caused numerical stability issues. Polynomial feature expansion gave us accuracy of 0.77613.

As mentioned in section II, we also have trained four separate models for each *PRI_jet_num*, this approach gave us final score of 0.804.

The figures 1 and 2 show our reasoning between choosing gamma parameter and avoiding regularization. The train/val loss/accuracy plots doesn't show any signs of overfitting, and interpreting these figures shows that optimal convergence occurs with *gamma* ≈ 0.25, after hand-tuning we chose value of 0.17 since it gave better accuracy.

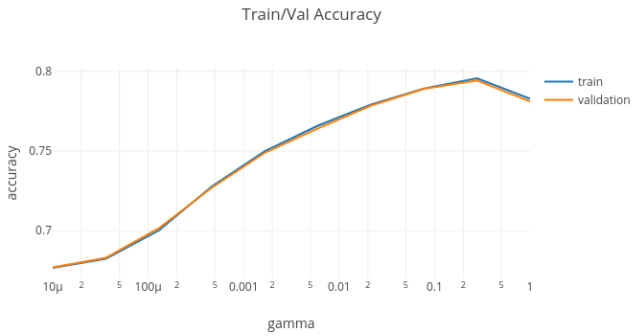

Fig. 2: Comparison of training and validation accuracy, with varying *gamma* and *n_iters* = 3000.

VI. Conclusion

In this report we described our approach for reaching 0.804 accuracy in *Higgs Boson Machine Learning Challenge*. We explained our reasoning behind choosing logistic regression with polynomial feature expansion. After implementing baseline model, several improvements were introduced, like mean subtraction, standard deviation normalization and missing data inputting. Finally, we showed systematic way to choose optimal hyperparameters for the algorithm.

REFERENCES

[1] Claire Adam-Bourdarios, Glen Cowan, Cecile Germain, Isabelle Guyon, Balazs Kegl, David Rousseau. *Learning to discover: the Higgs boson machine learning challenge.*

[2] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.*