CS-433 Machine Learning Project 1

Aryan Agal, Michael Chan, Vladislav Levitin

***Abstract*—The following study consists in binary classifying of a detection to the class of Higgs Boson/not Higgs Boson. Through application of linear models such as Logistic Regression with a loss function metrics of sigmoid of mean squares. Improvements of the direct applications boils down to fine-tuning of hyperparameters and a in-depth data pre-processing/ data augmentation.**

I. Introduction

The aim of this report will be to train a model that best classifies the Higgs Boson particle based on the 30 features given. We will be primarily focusing on training our model using mean squares and logistic regression classification models. Our prediction is that logistic regression with a n^{th} polynomial degree would give us the most accurate results, as it's output is binary classification. The polynomial of degree n is likely to range between 3 to 7, as going higher may cause over-fitting.

II. Establishing the Baselines

Our first goal is to establish baselines for each model. We started with every training method we had and tried to observe how the models perform with minimal preprocessing.

We begin with shuffling the data and splitting the data into training and validation in a 90:10 split. The data often had undefined/NaN values for some of the features of each data point. We resolved this issue by using the average of the data feature and replaced NaNs for each data-point, per feature.

We then ran the data through analytical least squares, and ridge regression, cycling over regularization paramater(λ) values from $1e^{-4}$ to $1e^{+4}$, multiplying by a factor of 10 each time. We achieved an accuracy of 74% with least squares and ridge regression for almost all λ values, for both train and test sets.

We then tried least squares GD and SGD with some trial initialization which gave us respectively 72.1% and 68.1% accuracy on validation set after 1000 iterations. Remains logistic regression and regularized logistic regression. Due to costly computation demand, we had to reduce the points set size that we used to direct ourselves i.e apply mini-batch. We obtained respectively 72.7% and 73.2%.

III. Data Pre-processing

To further increase the accuracy of our model we applied further data processing methods as removing the offsets, normalizing and feature augmentation the data. By augmenting the data with polynomial term we were able to get a model that fits the data point more accurately.

Filling NaNs. The data often had undefined/NaN values for some of the features of each data point. We resolved this issue by using the average of the data feature, which was undefined/NaN for the datapoint.

The features of the data have different ranges and therefore we decided to normalize it. Normalization will change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. [1]

$$z = \frac{x_n - \mu}{\sigma}$$

The nomalized is given by the z value, where σ is the standard deviation of x and μ is the mean of x. Hence the mean μ of z is 0 and the standard deviation σ of z is 1.

The data set had multiple outliers for the different features which could have deviated the training model. As a solution we cut of the data to between 5% and 95% of the data. Hence:

$$5\% \leq P(x) \leq 95\%$$

To create polynomial, exponential and logarithmics models for the data we decided to use feature augmentation

In order to deal with the over fitting of the model for the data we use regularization:

$$y_n = f(wx_n) + \lambda w^2$$

Where lambda is a constant. λ allows for noise to be introduced even though it stretches the loss on the training set.

Along the idea of of feature augmentation, we are looking for an interesting feature to add in. Our attempt consist in reordering the variance of the product of features. Adding the N highest variances as feature augmentation.

IV. Models and Methods

A. Analyitcal Least Squares/ Ridge Regression

As a baseline we first implemented the function for least squares and ridge regression, using the normal equations (note: GD are not used purposefully as noise can be simulated by Ridge and if we don't require noise for validation, then least squares would always upper bound). Since we were establishing the baseline we used minimal pre-processing by replacing the NaN's by the mean value of the feature. We trained on 80% of the training data and tested on remaining 20%. We achieved an accuracy of 75% on the 80% training data and 70% on the 20% on the training data. Deeper research improved even further the accuracy on validation sets. Combining multiple $\lambda \in [10^{-10}; 10^{7}]$ and degree for polynomial feature augmentation, we reached an accuracy of 82% when using $\lambda = 10^{-8}$ and degree:$= 14$.

[1] ¡://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029¿

B. Logistic Regression

In this part we employed the logistic regression. The minimization algorithm we used for the loss function was SGD. Multiple steps for training were done, and at each step different hyperparameters:

- $\gamma \in [10^{-4}; 10^{-7}]$
- max_iters$\in [|100; 30000|]$
- batch_size$\in [|64; 512|]$

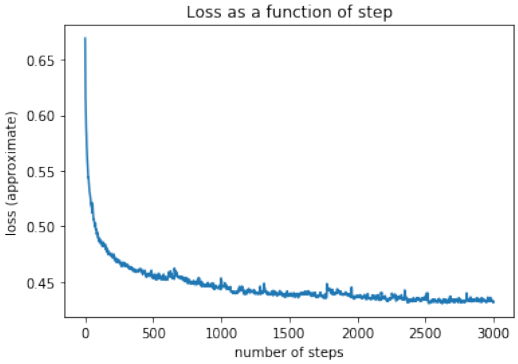
As a pre-training job, we did standardization of the datasets in order to get back to $\mu = 0$ and $\sigma = 1$. On top of this we feature augmented using polynomials of degree 6. (Interfeature augmentation TODO). The initial weights was determined by trial and error, and we ended up deciding that $|(W_0)_i| \leq 1$ was a fair starting point. The process of minimization, was done case by case in which we would increase the batch size and increase γ when we wanted to keep working on our current local minimum and decrease the batch size for higher noise to try other locality. We were keeping track of the descent through the loss per data point and as the loss encountered NaN issues due to negative log (computational issue) we had to modify slightly the loss function (even if it meant rounding it) in order to avoid NaNs (thus $|L_e - L_t| < \epsilon$) .

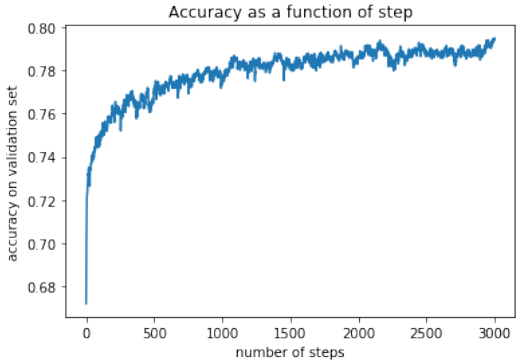
C. Regularized logistic regression

This time around for regularized logistic regression, we proceed the same way but moreover tried around different λ. The general trial is testing out for a hundred iterations with $\lambda \in [|10^{-5}; 10^{4}|]$

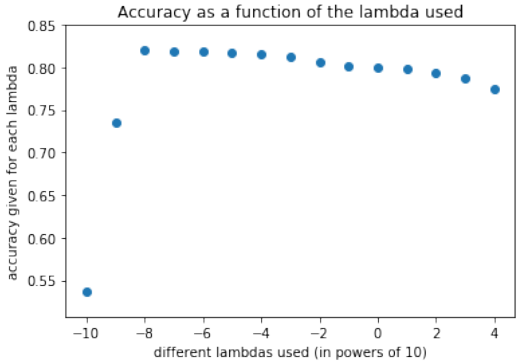
V. Results

We were able to get accurate results using logistic regression, as it is intended to be used for binary classification. Using regularized logistic regression we were able to tune the regularization parameter λ, to minimize the over fitting. The highest accuracy of the regularized logistic regression was using a 7th degree polynomial, 3000 iteration and regularization parameter $\lambda = 10^{-8}$ was 79%. The regularized logistic regression used the mini-batch algorithm with a batch size of 100 iterations. The data was also standardized in order to achieve more accurate results. The





For the analytical least squares we got around 75% on the validation data, just with filling the NaN's as data prepossessing. After we normalized the data and added polynomial features up to 14 degrees we were able to push the accuracy to 82% on the validation data. The following graph shows how the loss varies with the regularization parameter.



Observing the graph the optimal regularization parameter for ridge regression was $\lambda = 10^{-8}$.

VI. Discussion

The result of ridge regression producing the most accurate model was unexpected as it deviated from our prediction of logistic regression producing the most accurate model. Another unexpected finding was that using a polynomial of 14th degree on ridge regression produced the most accurate model on the validation data and did not over fit.

Using the regularized logistic regression we were able to quikly decrease the loss with mini batch, however after 500 iterations the reduction in loss became exponentially slower with it step and started fluctuating. This was likely due to that the learning rate γ was too large and the step size being too big to closer approach the minimum, or the algorithm approached a local minimum. We tried reducing the learning rate from 10^{-4} to 10^{-5}, however it became computationally too expensive and took to long to approach the minimum.

We attempted to use the interaction terms by cross multiplying the features to see using the relationship between different features would create a better model.[2] The model turned out to have 78% accuracy on the validation data. We added polynomial features up to 3rd degree to the data, but

[2]https://christophm.github.io/interpretable-ml-book/interaction.html

that only pushed the accuracy to 80%. Further increasing the degree was computationally to expensive.

VII. Summary