Machine Learning Project 1 : The Higgs Boson

Freundler Nicolas, Lanzrein Johan, Wicht Bruno
EPFL, Switzerland

***Abstract*—During this project, we apply the basic machine learning used in class in a real world situation. The problem to solve is the one of classifying particles as Higgs boson or not. We start by implementing the methods that we have seen in class and then implement other methods to model our dataset. Using feature expansion and a perceptron function, we get a model that can predict the particles with 82% correctness.**

I. Introduction

The Higgs boson is an elementary particle. We aim to build a precise and fast classifier for a given dataset from the CERN. Given this dataset, our main goal was to decide whether a given particle was a Higgs Boson or not. We start by visualizing the data to see if there are any outliers and if we can note some relation between features. Afterwards we proceed to clean the data and identify outliers in order to remove them. To classify the data, we first explore methods that have been discussed during class before moving on to more efficient ones . The most effective result being a perceptron.

II. Models and Methods

We first start by visualizing the data by using the seaborn library. This allows us to understand better how the data is distributed and also to see if there is any major trends for a given data point to be a Higgs-Boson. The visual library is very helpful and allows us to use a high level interface. Using this visualization we saw that for example feature 26 had a very different distribution depending if it was or not a Higgs boson. As the figure was too large for the report it is included in the zip file for reference if needed.

This lead us to add more features for columns where the data was only distributed between 0 and 1. Afterwards we turn ourselves to preprocessing the data. As we have seen while visualizing the set, there is a number of outliers on a few features. The outliers are the values -999. To deal with such values, we have decided to substitutes the -999 values with the mean of the feature without the outliers. With such a process, the outliers will be set to 0 when we normalize our data. Which brings us to our next step, normalizing the data. In this step, we do a normalization of all the features data in order to have all features with same mean and variance. Experimenting with various ideas, we decided to add a two dimensional feature expansion. To compute this we use this formula : for each pair of features, we add a new column containing their product.

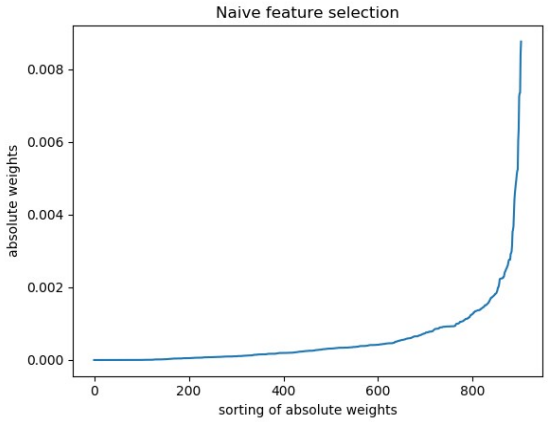
We tested the different methods seen in course, such as linear regression, polynomial regression (on the 2D-expanded data set), regularized polynomial regression and regularized logistic regression. Ridge-regularized linear and polynomial regression can be done in direct calculation or with (stochastic) gradient descent and logistic regression is computed only using (stochastic) gradient descent. We mainly used gradient descent for non expanded data, but stochastic gradient descent was sometimes preferable for memory management. Of course, we minimized loss functions corresponding to the theoretical basis of the method: least squares for linear and polynomial, categorical cross entropy for logistic regression. It is important to note that since these loss functions are not comparable, they were not considered for performance assessment: for this we used accuracy. Accuracy is simply true negatives and true positives, and is the measure of Kaggle competition. To compare the different models with different hyperparameters (different regularization), we performed a k-fold cross validation with conservation of label proportion.

We also tried a method that was not covered in this course: the single perceptron. The single perceptron is a single neuron that outputs 1 or -1 based on input vectors. Perceptrons are very similar to logistic regression in the sense that it performs a linear classification. Note that we used it with 2D-expanded data, so the classification is quadratic. We didn't perform any cross-validation for hyper-parameter tuning with this one. As regularization, we performed a bagging: generating several models with exactly same hyper-parameters (step size and iterations) and then calculating the mean of the models. Assuming that the models have similar performances, doing that will reduce their variances. Perceptron is designed for GD and SGD, and its update rule is similar to but simpler than logistic regression. The update rule can include a normalization of the weight vector, we chose L1 normalization. Perceptron's update rule :

$$w_{i+1} := w_i + \gamma(y - sign(x \cdot w_i))$$
$$w_{i+1} := \frac{w_{i+1}}{|w_{i+1}|}$$

The idea of feature selection arose quite early in the project. Theoretically, an exhaustive search of the best model would give the best feature combination, but it would lead to compare $2^{\#offeatures}$ models using cross-validation and this is far beyond our computational power. A wrapping method was implemented: in the first step, it selects the best feature, then fixes it tries to find another feature that, together with the precedent one, will lead to the best accuracy after cross-validation. Total operations lead to comparing $\frac{(\#features)(\#features+1)}{2}$. At the end, this results in as many models as features. This one was affordable with early models that contained at most 30 features. The third method to be implemented is called "naive"

feature selection. The underlying hypothesis of naive selection is that if some features are very important for a model, it will have a highest weight in classifier method (such as perceptron and logistic regression). It consists on training a model with all the features, sort the features in function of their absolute weights in order to select only a certain number of the heaviest one, then retrain. The advantage of this one is to be very quick.



Cross validation is a suitable way to estimate the test error of a given model. It is useful for model comparison and selection, especially when overfitting can occur. Overfitting is when a model is complex enough to learn the noise of the training data, has a great performance on this training data but is not optimal for new unseen data or test data . It was used for model comparison, but not systematically.

III. Results

While the standard methods produced satisfying results, we had to resort to more advanced method in order to get results above the 80% accuracy. Below is a table with the method we first tried and the results they yielded. Note that for those results we optimized over a range of parameters.

Method	Test acc.
Logistic regression, 100 iteration. learning rate 0.0019	0.65894
2Degree expansion,ridge regression lambda = 0.00001	0.79786
Same as before with a CV selected lambda of 0.0001	0.79799
Perceptron on 2Degree, bagged 30x, rate 10^{-6}	0.80985
Same as above with naive feature selection (200 heaviest)	0.82160
Same as above, 150 heaviest	0.82278
Bagged perceptron, with new expansion (905 features)	0.82207
Same as above, 450 heaviest	0.82298

With the cross-validation, we first used the ridge regression method and had results in the 80% range. We then used the perceptron method with more acurate results. As a final try, we tried to use the perceptron and train it with a cross-validation. However the results were not as effective as with the perceptron alone. So we discarded the results of this experience.

IV. Discussion

We notice that without a good feature cleaning, and a feature expansion, our predictions will be very weak. As shown in example 1 of the table above. The logistic regression, is barely capable of getting more than 70% test accuracy. This shows the importance of doing features expansion. Moreover, we noticed that once the feature expansion had been done. We would get better results by selecting only the heaviest features. The reason behind it is that the less important features create mostly noise that adds some irregularity to the model classifier.

V. Summary

We explored basic methods to try and solve a basic classification problems. Through this project, we have first used the methods studied in class. Then we explored how feature expansion can considerably improve the correctness of a model by having more features available and more precise results while losing some time for computations. Finally, we implement a perceptron with bagging. Using this method and the feature expansion we get a classifier with a precision of 82%. This result could be improved by doing a feature expansion on more dimensions, or by choosing better hyper-parameters.