Higgs Boson; To be, or Not to Be

Amir REZAIE, Arash ASKARI, Antonin VIANEY, EPFL, Lausanne, Switzerland

Due to rapid decay, observing Higgs Boson particles is not feasible. However, by examining the features recorded at the laboratory using machine learning algorithms, it is possible to detect them. A regression-based machine learning algorithm was developed in Python framework for the detection of Higgs Boson particles based on the features in the given database.

1. Getting Started

Some prerequisites shall be met before executing the code. You need a Python environment, an IDE, and several libraries of Python pre-installed before executing the code. You also have to download the project dataset before the execution.

1.1. Package Installation

1.1.1. Python

Python version 3.6.8 was used to develop this project. You can install the latest version of Python here.

1.1.2. IDE

You need an IDE to develop, debug, and execute the Python code. In the current project, Spyder 3.3.6 and PyCharm 2019.2.3 (Community Edition) were used. You can find the download instructions for the latest Spyder here. Moreover, the latest Community or Professional Edition of PyCharm can be downloaded through this link.

1.1.3. NumPy

NumPy library version 1.17.3 was used in this project.

1.1.4. CSV

CSV library version 1.0 was used in this project.

1.1.5. MatplotLib

MatplotLib library version 3.1.1 was used in this project.

1.2. File and Folder Structure

Three folders are available in the project structure.

The **data** folder contains the *train.csv* and *test.csv* files. These are the databases provided for training and testing the machine learning algorithm, respectively. The *final submission file* that is uploaded to the competition website is also generated in this folder.

The **report** folder contains a two-page pdf summary report of this project (*report.pdf*). Also, a copy of this *ReadMe.pdf* file is included in the folder.

The **src** folder contains all the Python code files. The function of the files is described in detail in the following sections. After executing the projects, an *accuracy.txt* file is generated in the folder, which stores the model accuracies for each parameter configuration of the machine learning algorithm.

2. Python Codes and Functions

A series of modules were developed to perform the requested procedures. Below, you can find the list of files and descriptions of their functionality.

2.1. Exploratory_data_analysis.ipynb

This Jupyter Notebook is mainly used for data visualization. After exploring the data and plotting the distributions, the outliers could be recognized. The outliers can deteriorate the accuracy of the model drastically. Therefore, removing them is one of the main procedures in data cleaning.

2.1.1. NaN Values

NaN values often appear in datasets as a result of no available recordings. The existence of *NaN* values can lead to sudden interruption of the code when dealing with the data. Therefore, in the beginning, the data was checked for *NaN* values.

2.1.2. Visual Inspection

To have a general view of the distribution of the data and to find any abnormal irregularities in the dataset, the provided features were examined using histograms and box plots.

2.1.3. Outliers

The distribution of the *signal/background* recordings were checked after the removal of outliers from the dataset. The imbalance of the number of *signal* and *background* recordings was also checked as it can lead to a bias in the predictive model.

2.2. test_implementations.ipynb

This Jupyter Notebook is developed to check the functions coded in *implementations.py*. The functions are checked if they generate correct output for simple known input.

Two toy examples are provided. The first toy example is defined to check the *least_squares*, *least_squares_GD*, *least_squares_SGD*, and *ridge regression* functions in the *implementations.py* file. The second toy example was developed to control the *logistic_regression* and *reg_logistic_regression* functions in *implementations.py*.

2.3. proj1_helpers.py

This Python file was provided in the project material. However, a number of functions were added to the file, described below.

2.3.1. box_plot & box_multi_plot & hist_multi_plot

These functions are utilized to visualize the data features. By exploring the data and its distributions, a general view of the samples is obtained to clean the data. The *box_plot* function is used to draw a box plot

of a selected feature. On the other hand, *box_multi_plot* and *hist_multi_plot* functions are utilized to present several numbers of box plots and histograms for various features.

2.3.2. divide_database

This function is utilized to divide the given database into four sub-groups according to their **PRI_jet_num** feature value.

2.3.3. count_categorical

After dividing the database according to their **PRI_jet_num** feature value, this function gives a quantitative view of the number of *signal* and *background* recordings for each sub-group.

2.3.4. cleaning_data

We have realized the distribution of the database features in previous steps. Therefore, several features were chosen to be deleted from the database. These are the features that cannot be a good classifier as they may have a constant distribution or contain lots of irrelevant values. Using the *cleaning_data* function, improper features are deleted from the database. Also, *-999 values* in **DER_mass_MMC** feature were replaced by the median value of the feature.

2.3.5. standardize

To diminish any side effects of having different scales in features, this function is used to standardize the database features.

2.3.6. PCA

This function is used to find the projection of features in principal directions.

2.3.7. build_poly & build_log & build_log_inverse & build_sqrt & differences & ratios & build_gaussian_basis

These are the functions used for feature augmentation. They create a new feature based on the original features and augment them to the database.

2.3.8. next_batch

This function creates standard mini-batch-size 1 for SGD algorithm as per suggestion in project instructions.

2.3.9. logistic_function

This function returns the sigmoid function for logistic regression.

2.3.10. predict_logistic

This function makes predictions based on the logistic regression algorithm.

2.3.11. distance_euclidean

The function calculates the Euclidean distance between two given vectors.

2.4. implementations.py

This Python file includes six machine learning functions required by the project instructions. All the functions return the last weight vector of the method and its corresponding loss value.

2.4.1. least_squares_GD

Function for linear regression using gradient descent.

2.4.2. least_squares_SGD

Function for linear regression using stochastic gradient descent.

2.4.3. least_squares

Function for least-squares regression using normal equations.

2.4.4. ridge_regression

Function for ridge regression using normal equations.

2.4.5. logistic_regression

Function for logistic regression using gradient descent or SGD.

2.4.6. reg_ logistic_regression

Function for regularized logistic regression using gradient descent or SGD.

2.5. loss.py

This Python file includes the loss functions that are utilized by the machine learning functions defined in *implementations.py*.

2.5.1. compute_loss_mse

Function for calculating the *mean-squared-error loss* for linear and ridge regression models.

2.5.2. compute_loss_logistic

Function for calculating the *cross-entropy loss* for logistic regression models.

2.6. gradient.py

This Python file includes the functions that calculate the gradients for the machine learning functions defined in *implementations.py*.

2.6.1. compute_gradient_GD

The function calculates the full gradient of the *mean-squared-error loss* function with respect to the vector of parameters for linear models.

2.6.2. compute_gradient_SGD

The function calculates the stochastic gradient of the *mean-squared-error loss* function with respect to one data point.

2.6.3. compute_gradient_logistic_GD

The function calculates the full gradient of the *cross-entropy loss* function in logistic regression problems.

2.6.4. compute_gradient_logiistic_SGD

The function calculates the stochastic gradient of the *cross-entropy loss* function with respect to one data point in logistic regression problems.

2.7. run_base_model_logistic.py

This Python file is the main executable of the project when the base logistic regression model is considered.

The file reads the train data and then changes the encoding of target values from **{-1, 1}** to **{0, 1}**. The change in encoding is due to the fact that logistic regression models work with 0 and 1 labels.

As mentioned in previous sections, a general survey of the data was performed by the authors. Therefore, at this stage, outliers identified at previous steps, are deleted from the dataset.

To be able to control the accuracy of the training process, the dataset is divided into *training* and *validation* sets by a predefined **ratio**.

```
ratio = 0.8
X_tr, y_tr, ids_tr, X_val, y_val, ids_val = split_data(X, y, ids, ratio, seed=5)
```

Then, both *training* and *validation* datasets are divided into four subgroups based on their **PRI_jet_num** feature value.

```
X_tr_0, y_tr_0, _ = divide_database(y_tr, X_tr,ids_tr,PRI_jet_num = 0)
X_tr_1, y_tr_1, _ = divide_database(y_tr, X_tr,ids_tr,PRI_jet_num = 1)
…
```

After that, for each group of data, the features that were not realized as good classifiers (for example: having a constant distribution of values through their corresponding range) were deleted from the *training* and *validation* datasets.

```
…
X_tr_2,y_tr_2 = cleaning_data(y_tr_2, X_tr_2, irrelevant_feature_columns=[22], replace_missing_data = True)
X_tr_3,y_tr_3 = cleaning_data(y_tr_3, X_tr_3, irrelevant_feature_columns=[22], replace_missing_data = True)
…
```

Also, to avoid any biasing in the predictive model, the features with high respective correlations were also removed.

```
X_val_0,_ = cleaning_data(y_val_0, X_val_0, [3], replace_missing_data = True)
```

Then to avoid any scaling problem for the data features, all the features were standardized.

```
X_val_0,_ = cleaning_data(y_val_0, X_val_0, [3], replace_missing_data = True)
```

After that, the intercept term was added to the features, and then the predictive model parameters and variables were initiated.

To tune the predictive model parameters the model is looped through a list of parameter values. Validation accuracy is used to select the best model parameters. The model accuracies corresponding to each parameter are stored in *accuracy.txt* file in the **src** directory.

After the completion of training, test data is read and divided into four subgroups according to the **PRI_jet_num** feature values. The same cleaning, standardization, and interception procedures were performed on the test data. Finally, the data was fed into the predictive model and predictions were made.

2.8. run_augmented_model_xx_logistic.py

This Python file is the main executable of the project when the augmented logistic regression model is considered. The '**xx**' can be from '**01**' to '**06**'. The '**01**' augmented model considers *Gaussian basis function*, '**02**' considers *square root*, '**03**' considers *difference*, '**04**' considers *ratio*, '**05**' considers *logarithm* and '**06**' considers *inverse of logarithm* augmentations. The augmentations are applied to increase the nonlinearity of the model.

The structure of the file looks the same as the structure of the *run_base_model_logistic.py* file. The only two differences are 1) There would be a nested loop for selecting the best number of basis functions, and 2) The data is augmented after standardization.

```
best_basis_0 = basis_list[np.where(val_loss_0 == np.min(val_loss_0))[1][0]]
best_basis_1 = basis_list[np.where(val_loss_1 == np.min(val_loss_1))[1][0]]
…
```

2.9. run.py

This Python file is the main executable of the project which leads to the best prediction results. The structure of the file is very much the same as the structure of the *run_augmented_model_01_logistic.py* file. Only bagging and majority voting are applied through a loop so that the algorithm can find the best results when a large number of models are generated for a small part of the dataset. Also, the datasets are divided further into two groups according to their **DER_mass_MMC** feature values (*no_f1*: the data that has a **DER_mass_MMC** value of **-999** and *with_f1*: the rest of the data).

2.10. proj1_helpers_run.py

This Python file only provides two functions required by the *run.py* for reproducing the best model.

2.10.1. sub_group

This function is used for the further grouping of the datasets based on their **DER_mass_MMC** feature values.

2.10.2. fit

The function is used to train the best model for the given training dataset, considering the best model parameters.

3. Reproducibility and Execution

Due to the limitations of upload files, the train and test datasets were not included in the zip package. Therefore, to execute the models, please **first** download the datasets (*train.csv* and *test.csv*) and place them in the **data** folder. Then, please note that the working directory of your IDE shall be set as the **src** folder.

After that, to execute the models, you can just run the *run.py*, *run_base_model_logistic.py* or *run_augmented_model_xx_logistic.py* scripts. The output submission files will be made in the **data** directory. Please note that: **To run the best model, you should execute the *run.py*.**