README : Searching for the Boson

Ducommun Colin, Elelamy Rayan, Van de Velde Anne-Sophie

1 Prerequisites

The code is written in Python3, and makes extensive use of Numpy, the fundamental package for scientific computing in Python. Another external module used is `csv` - it intervenes in importing ghe flow of data to and from the Python environment, in order to read the data from a `.csv` file and to write the predictions in a `.csv` file respectively. Therefore the user must have both packages to run this code. For installation information, please refer to the Python documentation.

2 Executing the code

Extract the `.zip` and open the terminal at the location of the extracted files. Then enter: `python.run.py`. The file `run.py` contains an executable Python script that takes train data to compute a model. To see more details about the objective of this scrpit, one should read our report. The desired train and tests sets are to be put in a `train.csv` and `test.csv`, which are provided as placeholders.

3 Description of the code

The script first loads the data and initializes the hyperparameters. The values provided initially coincide with the optimal values mentioned in the report.

First missing values are treated by deleting the columns where they occur. Then the script splits the training and test data into 3 categories according to PRI_jet_num (see justification in the report). Then, a polynomial extension up ti the value of "degree" fallows. Finally, pairwise products of some if the features are performed. After this treatment, data is normalized and put in an appropriate format for the learning process.

The initial weights w are random. The training is done using a regularized logistic regression, with parameter lambda. The optimal w is approached via an iterative method: the gradient descent (see justification in report). The method outputs the optimal weights w and the loss that corresponds to it.

Then, the test data goes through the treatment described above: from replacing missing values, to standardization. The predictions are performed then converted to the appropriate format $\{-1, 1\}$ according to the best threshold (found via Kaggle). This process is repeated for each category of PRI_jet_num values $(0, 1, 2, 3)$. The predictions are then rearranged to match the order of the original sample and put out using the csv module. They can be found in the same folder as the run.py script.

4 implementations.py

This file contains the methods that were to be implemented :

i) least_squares_GD(y, tx, initial_w , max_iters, gamma) : performs a linear regression using a gradient descent algorithm.

ii) least_squares_SGD(y, tx, initial_w , batch_size , max_iters, gamma) : performs a linear regression using a stochastic gradient descent algorithm

iii) least_squares (y, tx) : performs a least squares regression using normal equations.

iv) ridge_regression (y, tx, lambda_) : performs a ridge regression using normal equations.

v) logistic_regression (y, tx, initial_w , max_iters, gamma) : performs a logistic regression using gradient descent.

vi) reg_logistic_regression (y, tx, initial_w , max_iters, gamma, lambda_) : performs la regularized logistic regression using a gradient descent.

The inputs, when present, are :

- y which is a vector of flags (here it is an array of -1's and 1's);
- tx which is the matrix of features (in an order corresponding to y);
- initial_w which is the starting weight for the iterative methods;
- max_iters which sets the maximum number of iterations for the iterative methods;
- gamma which is the step size for the iterative methods;
- lamda_ which is the factor of penalization in the ridge regressions;

All of these methods return the predicted weights w and the loss. They are to be used cautiously and basic knowledge in machine learning is assumed.

5 utilities.py