There are three python files needed to get the desired system ML, implementations.py, project1_helpers.py and run.py . In this readme we will describe all the functions defined in the implementations.py and project1_helpers.py files. And we will explain the different steps of run.py to get the desired predictions.

Implementations.py :

In this python file, we have used only one external library, the numpy library. We have 8 functions defined in this file. We have 8 functions defined in this file which will be the methods we will use to estimate our model.

---

least_square_GD(y, tx, w_init, max_iters, gamma)

Apply the ordinary least squares regression with gradient descent algorithm.

Parameters:

- y : numpy_array which represents the response variables.
- tx : numpy_array which represents the design matrix, each column is a feature/ explanatory variable.
- w_init : numpy with represents our initial parameters for our gradient descent, usually taken as zero.
- max_iters: integer which represents the maximal number of iterations of gradient descent.
- gamma : positive real number which represents the step-size of gradient descent.

Returns: weights, loss (numpy_array, real number)

Returns the weights associated with the ordinary least squares method and the associated loss, optimized by gradient descent algorithm.

---

least_square_SGD(y, tx, w_init, max_iters, gamma)

Apply the ordinary least squares regression with stochastic gradient descent algorithm. In particular, it uses the standard mini-batch-size 1 by uniformly choosing a datapoint with replacement at each iteration.

Parameters:

- y : numpy_array which represents the response variables.
- tx : numpy_array which represents the design matrix, each column is a feature/ explanatory variable.
- w_init : numpy with represents our initial parameters for our gradient descent, usually taken as zero.
- max_iters: integer which represents the maximal number of iterations of gradient descent.
- gamma : positive real number which represents the step-size of gradient descent.

Returns: weights, loss (numpy_array, real number)

Returns the weights associated with the ordinary least squares method and the associated loss, optimized by stochastic gradient descent algorithm.

---

ridge_regression (y, tx, lambda_)

Apply the ridge regression by using the normal equations.

Parameters:

- y : numpy_array which represents the response variables
- tx : numpy_array which represents the design matrix, each column is a feature/ explanatory variable
- lambda_ : positive real number which is a hyperparameter of the model, and control the amount of shrinkage of parameters of the model. The larger this number is, the more the parameters are forced to be close to zero. This parameter also allows to regularize the model when the design matrix has multicollinearity problems, in particular it allows to perform the inversion of the final matrix to obtain the corresponding parameters.

Returns: weights, loss (numpy_array, real number)

Returns the weights associated with ordinary least squares method and the associated loss, obtained by normal equations.

---

least_squares (y, tx)

Apply the ordinary least squares method using the normal equations. Equivalently, it is a ridge regression when the hyperparameter lambda_ is equal to 0.

Parameters:

- y : numpy_array which represents the response variables.
- tx : numpy_array which represents the design matrix, each column is a feature/ explanatory variable.

Returns: weights, loss (numpy_array, real number)

Returns the weights the weights associated with the ordinary least squares and the associated loss, obtained by normal equations.

---

reg_logistic_regression(y, tx, lamba_, w_init, max_iters, gamma)

Apply the regularized logistic regression with gradient descent algorithm.

Parameters:

- y : numpy_array which represents the response variables.
- tx : numpy_array which represents the design matrix, each column is a feature/ explanatory variable.
- lambda_ : positive real number which is a hyperparameter of the model, and control the amount of shrinkage of parameters of the model. The larger this number is, the more the parameters are forced to be close to zero. This parameter also allows to

regularize the model when the design matrix has multicollinearity problems, in particular it allows to perform the inversion of the final matrix to obtain the corresponding parameters.

- w_init : numpy with represents our initial parameters for our gradient descent, usually taken as zero.
- max_iters: integer which represents the maximal number of iterations of gradient descent.
- gamma : positive real number which represents the step-size of gradient descent.

Returns: weights, loss (numpy_array, real number)

Returns the weights associated with regularized logistic regression method and the associated, optimized by gradient descent algorithm.

---

logistic_regression(y, tx, w_init, max_iters, gamma)

Apply the regularized logistic regression with gradient descent algorithm. Equivalently, it is a ridge regression when the hyperparameter lambda is equal to 0.

Parameters:

- y : numpy_array which represents the response variables.
- tx : numpy_array which represents the design matrix, each column is a feature/ explanatory variable.
- w_init : numpy with represents our initial parameters for our gradient descent, usually taken as zero.
- max_iters: integer which represents the maximal number of iterations of gradient descent.
- gamma : positive real number which represents the step-size of gradient descent.

Returns: weights, loss (numpy_array, real number)

Returns the weights associated with logistic regression method and the associated, optimized by gradient descent algorithm.

---

reg_logistic_regressionSGD(y, tx, lamba_, w_init, max_iters, gamma)

Apply the regularized logistic regression with stochastic gradient descent algorithm.

Parameters:

- y : numpy_array which represents the response variables.
- tx : numpy_array which represents the design matrix, each column is a feature/ explanatory variable.
- lambda_ : positive real number which is a hyperparameter of the model, and control the amount of shrinkage of parameters of the model. The larger this number is, the more the parameters are forced to be close to zero. This parameter also allows to regularize the model when the design matrix has multicollinearity problems, in particular it allows to perform the inversion of the final matrix to obtain the corresponding parameters.

- w_init : numpy with represents our initial parameters for our gradient descent, usually taken as zero.
- max_iters: integer which represents the maximal number of iterations of gradient descent.
- gamma : positive real number which represents the step-size of gradient descent.

Returns: weights, loss (numpy_array, real number)

Returns the weights associated with regularized logistic regression method and the associated, optimized by stochastic gradient descent algorithm.

---

logistic_regressionSGD(y, tx, w_init, max_iters, gamma)

Apply the regularized logistic regression with stochastic gradient descent algorithm. Equivalently, it is a ridge regression when the hyperparameter lambda is equal to 0.

Parameters:

- y : numpy_array which represents the response variables.
- tx : numpy_array which represents the design matrix, each column is a feature/ explanatory variable.
- w_init : numpy with represents our initial parameters for our gradient descent, usually taken as zero.
- max_iters: integer which represents the maximal number of iterations of gradient descent.
- gamma : positive real number which represents the step-size of gradient descent.

Returns: weights, loss (numpy_array, real number)

Returns the weights associated with logistic regression method and the associated, optimized by stochastic gradient descent algorithm.

---

proj1_helpers.py :

In this python file, only the external packages numpy and csv are used. The 3 functions load_csv_data, predict_labels, create_csv_submission were provided at the beginning of the project. The others functions have been defined in order to process the data.

---

load_csv_data(data_path, tx, sub_sample=False)

Loads data.

Parameters:

- data_path: path to the file to read.

- sub_sample : bool, if false return the entire data in the corresponding file, if true return a sub-sample of the data.

Returns: yb, input_data, ids (numpy_array, numpy_array, integer)
Returns y (class labels), tx (features) and ids (event ids)

---

predict_labels(weights, data)
Predicts the labels of the response variable which take value -1 or 1, by using a criterion of the sign, i.e. the label of a response variable is given by the sign of his estimated value by the model which is obtained by ordinary least squares method.

Parameters:
- weights: numpy_array which represents the weights obtained by using the least squares regression.
- data : numpy_array which represents the design matrix, each column is a feature/ explanatory variable, for which we want to predict each row.

Returns: y_pred (numpy)
Returns the predictions of the answer variables corresponding to data.

---

create_csv_submission(ids, y_pred, name)
Generate an output file in .csv format for submission to Kaggle or AIcrowd

Parameters:
- ids: numpy_array which represents event ids associated with each prediction.
- data : numpy_array which represents predicted class labels.
- name : string name of .csv output file to be created.

Returns:

Standardize(X):
Standardizes columns of design matrix X such that each column has mean 0 and variance 1.

Parameters:
- X : numpy_array which represents the design matrix X.

Returns: std_X (numpy_array)
Returns standardized matrix X.

---

meaningless_value_mean(X):
Replaces undefined value -999 by the mean of the features in design matrix X.

Parameters:

- X : numpy_array which represents the design matrix X.

Returns: X (numpy_array)

Returns matrix X with meaningless values replaced.

---

meaningless_value_median(X):

Replaces undefined value -999 by the median of the features in design matrix X

Parameters:

- X : numpy_array which represents the design matrix X

Returns: X (numpy_array)

Returns matrix X with meaningless value replaced

---

predict_log(weights, data)

Predicts the labels of the response variable which take value 0 or 1, by using a criterion of threshold of the probability at 0.5 , i.e. the label of a response variable depends on the fact that the event y=1 occurs with a probability greater than 0.5, probability modeled by the logistic regression

Parameters:

- weights: numpy_array which represents the weights obtained by using logistic regression method.
- data : numpy_array which represents the design matrix, each column is a feature/ explanatory variable, for which we want to predict each row

Returns: y_pred (numpy)

Returns the predictions of the answer variables corresponding to data.

Sub_array_Generator (y, tx, jet_nbr, bool_mass):

Provides data of a sub-problem defined by a particular jet number and whether or not the mass is defined or undefined.

Parameters:

- y : numpy_array which represents the response variables.
- tx : numpy_array which represents the design matrix, each column is a feature/ explanatory variable.
- Ids: numpy_array which represents event ids associated with y
- *jet_nbr* : non negative integer between 0 and 3 which represents the jet value of the the particular sub-problem to extract from the data
- bool_mass: binary value with value 1, if the sub-problem to extract has a defined mass and 0 otherwise

Returns: y_sub (numpy_array), tx_sub (numpy_array), id_sub (numpy_array)

Returns the output vector y, the feature matrix X and the id's of the experiment of the sub-problem to be extracted.

Feature_remover(tx_sub, jet_nbr, bool_mass, ill_defined_feature_tensor):

For the given sub-problem data (provided by the function *Sub_array_Generator*), it removes all undefined features.

Parameters:

- tx_sub : numpy_array which represents the sub design matrix, each column is a feature/ explanatory variable, for the sub-problem defined by a particular jet number and whether the mass is defined or not.
- *jet_nbr* : non negative integer between 0 and 3 which represents the jet value of the the particular sub-problem to extract from the data
- bool_mass: binary value with value 1, if the sub-problem to extract has a defined mass and 0 otherwise
- *ill_defined_feature_tensor:* numpy_array containing which features are ill-defined for a given sub-problem defined by a particular jet number and whether the mass is defined or not.

Returns, tx_sub_new (numpy_array)

The feature matrix X of the sub-problem to be extracted without the undefined values

---

Sub_problem_Generator(y,tx, ids,jet_nbr, bool_mass, ill_defined_feature_tensor):

For a given jet number and whether the mass is defined or not, this function creates the sub-problem data with all features removed.

Parameters:

- y : numpy_array which represents the response variables.
- tx : numpy_array which represents the design matrix, each column is a feature/ explanatory variable, for the sub-problem defined by a particular jet number and whether the mass is defined or not.
- Ids : numpy_array which represents event ids associated with y
- *jet_nbr* : non negative integer between 0 and 3 which represents the jet value of the the particular sub-problem to extract from the data
- bool_mass: binary value with value 1, if the sub-problem to extract has a defined mass and 0 otherwise
- *ill_defined_feature_tensor:* numpy_array containing which features are ill-defined for a given sub-problem defined by a particular jet number and whether the mass is defined or not.

return y_sub, tx_sub, id_sub

Returns the output vector y, the feature matrix X and the id's of the experiment of the sub-problem to be extracted (without the undefined features).

run.py :

The file run.py allows to compute the predictions of the model we choose. We import the functions defined in implementations.py, proj1_helpers.py and the numpy library. Run.py is divided in three steps, first we process the data, then we apply the method and finally apply the weights for

1) *Processing data*:

First of all, we load the data. Then we process the insignificant values of the covariates. For this, we propose several methods, the first method is the imputation, that is to replace the missing values (-999) by other values. The three possibilities of treatment belonging to this class of methods we have considered are the replacement of the undefined values by either 0, or by the average of the corresponding column or by the median of the corresponding column. The other method is to subdivide the main problem into 8 sub-problems depending on the values defined.

Once a method is chosen, let's suppose for example that the method you choose is the subdivision method, then in the code each time there is a line in the comment "If subdivision method used" you must put the next corresponding lines outside the comments and on the other hand when there is a comment "if imputation method used" you must put all the corresponding lines in the comments.

Next, we standardize the data. Then, we can add a square of the covariates or other powers of the covariates (using np.power(tx,d)) to obtain a polynomial regression and increase the complexity of the problem in order to improve our prediction. We standardize again the data to standardize the polynomial terms (perhaps it would have been better to standardize only once after adding the polynomial terms, but we could not test this fact, our tests were done only with a double standardization). Then, we add an intercept to the features, i.e. a column of ones. Then we initialize the parameters w_init, gamma, max_iters. We perform a grid search on the lambda to choose the model that makes the least error on the train set.

Throughout the code, you must adapt the comment lines according to the chosen data processing method.

2) *Method:*

First, we initialize the parameters w_init, gamma, max_iters. If you want to use the least_square method, you must adapt the corresponding comment lines in the code. It is better to use the version with the stochastic gradient descent for efficiency reasons.

If you want to use the regularized method, you must define the list of lambdas you want and adapt the corresponding lines or comments in the code. It is better to use the version with the stochastic gradient descent for efficiency reasons.

3) *Application on the test set:*

Adapt the lines and comments of code according to the choices made previously.

4) *Generate the same .csv file as the one on AIcrowd:*
Just execute run.py without making any changes, making sure that the files implementations.py, project1_helpers.py and the data are in the same directory.

Note: Maplotlib is imported to make graphs.