ReadMe for CS-433 Project 1

David Howard Neill Nathan Simonis Ludvig Warnberg Gerdin

Contents

We have submitted to you a folder containing three scripts written in python as well as a sub-folder containing the data you have provided us.

Summary.

In this section we give you a brief summary of each file we have submitted.

implementations.py

This script contains exclusively functions that have been used throughout the project. The 6 basic methods that implement the machine learning techniques seen in the courses are included. In addition, we have included other sections where the helpers needed to apply these techniques are provided. Among others, there are sections that calculate cost functions, gradients or allow us to import data. Each section is appropriately titled and the methods that make it up are adequately commented and described.

run.py

This script uses the functions present in the previous section in order to load the data as you provided them to us, clean them, prepare the explanatory variables and perform feature augmentation, train the model, make predictions on unseen data and finally, generate a submission file that is the same as the one we used to obtain the best accuracy score on the AICrowd platform.

cross_validation.py

This script, reproduces the procedure we used to select the parameters of our model. Namely, K-fold cross-validation using a grid search algorithm to select the parameters that allow us to obtain the best average accuracy in cross-validation. Please note that we have used a fairly large grid and therefore this script takes time to complete. At the end of it, it generates a text file with the final selected parameters that were used to generate the predictions.

data folder

This folder contains .csv files. This includes, the data that you have provided us with as well as the predictions file that was used on the AICrowd platform.

Description of the functions.

In this section, we will describe all the functions present in each of the scripts. We have included the name of the function, a description of what it does, the input variables and the output variables. For the input and output variables, we have given a description of what they represent as well as their data type.

implementations.py

Models

1.) least_squares_GD(y, tx, initial_w, max_iters, gamma, verbose=False):

```
Least squares with MSE loss and Gradient Descent.
Parameters
----------
y : Vector
    Dependent variable.
tx : Matrix
    Explanatory variables.
initial_w : Vector
    Initial weights.
max_iters : Integer scalar
    Maximum number of iterations.
gamma : Real scalar
    Learning rate.
verbose : Boolean, optional
    Print each step of Gradient Descent or not. The default is False.

Returns
-------
W : Vector
    Final vector of weights.
loss : Real scalar
    Loss given final weights.
```

2.) least_squares_SGD(y, tx, initial_w, max_iters, gamma, batch_size=1, verbose=False):

```
Least squares with MSE loss and Stochasitc Gradient Descent.
Parameters
----------
y : Vector
```

```
    Dependent variable.
tx : Matrix
    Explanatory variables.
initial_w : Vector
    Initial weights.
max_iters : Integer scalar
    Maximum number of iterations.
gamma : Real scalar
    Learning rate.
batch_size : Integer sclar, optional
    Size of the batch.  The default is 1.
verbose : Boolean, optional
    Print each step of Gradient Descent or not. The default is False.

Returns
-------
W : Vector
    Final vector of weights.
loss : Real scalar
    Loss given final weights.
```

3.) least_squares(y, tx):

```
Linear regression fit using normal equations.

Parameters
----------
y : Vector
    Dependent variable.
tx : Matrix
    Explanatory variables.

Returns
-------
W : Vector
    Final vector of weights.
loss : Real scalar
    Loss given final weights.
```

4.) ridge_regression(y, tx, lambda_):

```
Ridge regression fit using normal equations.

Parameters
----------
y : Vector
    Dependent variable.
tx : Matrix
    Explanatory variables.
lambda_: Real scalar
    Regularization parameter.

Returns
-------
W : Vector
```

```
    Final vector of weights.
loss : Real scalar
    Loss given final weights.
```

5.) logistic_regression(y, tx, initial_w, max_iters, gamma, verbose=False):

```
Logistic regression with log loss and Gradient Descent.
Parameters
----------
y : Vector
    Dependent variable.
tx : Matrix
    Explanatory variables.
initial_w : Vector
    Initial weights.
max_iters : Integer scalar
    Maximum number of iterations.
gamma : Real scalar
    Learning rate.
verbose : Boolean, optional
    Print each step of Gradient Descent or not. The default is False.

Returns
-------
W : Vector
    Final vector of weights.
loss : Real scalar
    Loss given final weights.
```

6.) reg_logistic_regression(y, tx, lambda_, reg, initial_w, max_iters, gamma, verbose=False, early_stopping=True, tol=0.0001, patience=5):

```
Regularized logistic regression with log loss and Gradient Descent with early stopping

Parameters
----------
y : Vector
    Dependent variable..
tx : Matrix
    Explanatory variables.
lambda_ : Real scalar
    Regularization parameter.
reg : Integer scalar
    L1 or L2 regularization.
initial_w : Vector
    Initial weights.
max_iters : Integer scalar
    Maximum number of iterations.
gamma : Real scalar
    Learning rate.
verbose : Boolean, optional
    Print each step of Gradient Descent or not. The default is False.
early_stopping : Boolean, optional
    Enable early stopping or not. The default is True.
```

```
tol : Real scalar, optional
    Minimum amount loss needs to change by. The default is 0.0001.
patience : TYPE, optional
    Number of iterations where there must be a decrease in loss by tol. The default is 5.

Returns
-------
W : Vector
    Final vector of weights.
loss : Real scalar
    Loss given final weights.
```

Cost functions

1.) mse(y, tx, w):

```
Mean squared error loss function.
Parameters
----------
y : Vector
    Dependent variable.
tx : Matrix
    Explanatory variables.
w : Vector
    Weights.

Returns
-------
loss : Real scalar
    MSE loss.
```

2.) logistic_error(y, tx, w):

```
Log loss function.
Parameters
----------
y : Vector
    Dependent variable.
tx : Matrix
    Explanatory variables.
w : Vector
    Weights.

Returns
-------
loss : Real scalar
    Log loss.
```

3.) reg_logistic_error(y, tx, w, lambda_, reg):

```
Log loss function with regularization term.
Parameters
----------
y : Vector
```

```
    Dependent variable.
tx : Matrix
    Explanatory variables.
w : Vector
    Weights.
lambda_ : Real scalar
    Regularization parameter
reg : Integer scalar
    L1 or L2 regularization
Returns
-------
loss : Real scalar
    Log loss with regularization term.
```

Gradients

1.) mse_grad(y, tx, w):

```
Compute gradient for MSE loss.
Parameters
----------
y : Vector
    Dependent variable.
tx : Matrix
    Explanatory variables.
w : Vector
    Weights.

Returns
-------
gradient : Real scalar
    MSE gradient.
```

2.) logistic_grad(y, tx, w):

```
Compute gradient for log loss.
Parameters
----------
y : Vector
    Dependent variable.
tx : Matrix
    Explanatory variables.
w : Vector
    Weights.

Returns
-------
gradient : Real scalar
    Log loss gradient.
```

3.) reg_logistic_grad(y, tx, w, lambda_, reg):

```
Compute gradient for log loss with regularization term.
Parameters
----------
```

```
y : Vector
    Dependent variable.
tx : Matrix
    Explanatory variables.
w : Vector
    Weights.
lambda_ : Real scalar
    Regularization parameter
reg : Integer scalar
    L1 or L2 regularization
Returns
-------
gradient : Real scalar
    Log loss with regularization term gradient.
```

Activation functions

1.) sigmoid(x):

```
Compute sigmoid function

Parameters
----------
x : Vector, Matrix or scalar
    Explanatory variables.
Returns
-------
sigmoid : Vector, Matrix or scalar
    Sigmoid function applied to input.
```

Helpers

1.) batch_iter(y, tx, batch_size, num_batches=1, shuffle=True):

```
Generate a minibatch iterator for a dataset.
Parameters
----------
y : Vector
    Dependent variable.
tx : Matrix
    Explanatory variables.
batch_size : Integer scalar
    Size of the batch.
num_batches : Integer scalar, optional
    Number of batches. The default is 1.
shuffle : Boolean, optional
    Shuffle data or not. The default is True.
Yields
------
y : Vector
    Mini-batch of y
```

```
tx : Matrix
    Mini-batch of tx
```

2.) import_data(path="data/"):

```
Import csv files of train and test data.  They must be in the same folder.
Parameters
----------
path : String, optional
    Directory of the files The default is "data/".
Returns
-------
train : Matrix
    Training set.
test : Matrix
    Testing set.
col_names : Vector
    Column names.
```

3.) create_csv_submission(ids, y_pred, name):

```
Creates an output file in csv format for submission to AIcrowd
Parameters
----------
ids : Vector
    Event ids associated with each prediction.
y_pred : Vector
    Predicted class labels.
name : String
    String name of .csv output file to be created.
```

4.) standardize_numpy(x, mean=None, std=None):

```
Standardize the original data set.
Parameters
----------
x : Matrix
    Data to standardize.
mean : Vector, optional
    Previously computed mean. The default is None.
std : Vector, optional
    Previously computed standard deviation. The default is None.
```

Prepare features

1.) split_X_y(train, test, cols):

```
Create tx matrix for train & test + y vector for train.
Parameters
----------
train : Matrix
    Training set.
test : Matrix
    Testing set.
cols : Vector
```

```
    Column names.

Returns
-------
tx_train : Matrix
    tx for training set.
y_train : Vector
    y for training set.
tx_test : Matrix
    tx for testing set.
```

2.) build_poly(x, degree):

```
Polynomial basis functions for each column of x, for j=1 up to j=degree, and single constant term.
Parameters
----------
x : Matrix
    Matrix to apply augmentation on.
degree : Integer scalar
    $j^{th}$ degree for polynomial basis function.

Returns
-------
phi : Matrix
    Augmented dataset.
```

3.) prepare_features(tx_nan, degree, mean_nan=None, mean=None, std=None):

```
Clean and prepare for learning.  Mean imputing, missing value indicator, standardize.
Parameters
----------
tx_nan : Matrix
    Explanatory variables.
degree : Integer scalar
    $j^{th}$ degree for polynomial basis function.
mean_nan : Vector, optional
    Compute column means with if necessary, The default is None.
mean : Vector, optional
    Previously computed mean. The default is None.
std : Vector, optional
    Previously computed standard deviation. The default is None.

Returns
-------
tx : Matrix
    Explanatory variables.
mean : Vector
    Mean of columns.
std : Vector
    Standard deviation of columns.
mean_nan : Vector
    Mean for columns with nan.
nan_cols : Vector
    Nan indicator columns.
```

Performance metrics

1.) logistic_prediction(tx, w):

```
Make a prediction with logistic regression model.
Parameters
----------
tx : Matrix
    Explanatory variables.
w : Vector
    Weights.

Returns
-------
y_pred : Vector
    Predictions.
```

2.) regression_prediction(tx, w):

```
Make a prediction with linear regression model.
Parameters
----------
tx : Matrix
    Explanatory variables.
w : Vector
    Weights.

Returns
-------
y_pred : Vector
    Predictions.
```

3.) f1_score(y_targ, y_pred):

```
Compute the F1 score of a prediction.
Parameters
----------
y_targ : Vector
    Dependent variable.
y_pred : Vector
    Prediction.

Returns
-------
score : Real scalar
    Score.
```

4.) accuracy(y_targ, y_pred):

```
Compute the accuracy of a prediction.
Parameters
----------
y_targ : Vector
    Dependent variable.
y_pred : Vector
    Prediction.
```

```
Returns
-------
score : Real scalar
    Score.
```

cross_validation.py

1.) cross_validation(y_tr, tx_tr, y_te, tx_te, comb, verbose=2):

```
Train model, compute in-sample and out-of-sample loss and accuracy.
Parameters
----------
y_tr : Vector
    Dependent variable in training set.
tx_tr : Matrix
    Explanatory variables in training set.
y_te : Vector
    Dependent variable in testing set.
tx_te : Matrix
    Explanatory variables in testing set.
comb : Dictionnary
    Combination of parameters.
    Example : {"gamma":0.1, "lambda":0.01, "reg":2}
verbose : Boolean, optional
    Print progress or not. The default is 2.

Returns
-------
loss_tr : Real scalar
    Training loss.
loss_te : Real scalar
    Testing loss
f1 : Real scalar
    Testing F1 score.
acc : Real scalar
    Testing accuracy.
```

2.) model_selection(y, tx, k_fold, degree, grid, seed, verbose=2):

```
Select the best model from all possible combinations of grid.
Parameters
----------
y : Vector
    Dependent variable.
tx : Matrix
    Explanatory variables.
k_fold : Integer scalar
    Number of folds for cross-validation
degree : Integer scalar
    $j^{th}$ degree for polynomial basis function.
grid : Dictionnary
    Set of all possible combinations.
seed : Integer scalar
```

```
    Random seed.
verbose : Boolean, optional
    Print progress or not. The default is 2.

Returns
-------
params : Dictionnary
    Best parameters for given grid.
```

3.) build_k_indices(y, k_fold, seed):

```
Build k indices for k-fold.
Parameters
----------
y : Vector
    Dependent variable.
k_fold : Integer scalar
    Number of folds for cross-validation.
seed : Integer scalar
    Random seed.

Returns
-------
k_indices : Matrix
    k indices for K-fold.
```

4.) prepare_split_data(y, tx, degree, k_fold, seed):

```
Split the dataset based on k-fold cross validation and prepare features.
Parameters
----------
y : Vector
    Dependent variable.
tx : Matrix
    Explanatory variables.
degree : Integer scalar
    $j^{th}$ degree for polynomial basis function.
k_fold : Integer scalar
    Number of folds for cross-validation.
seed : Integer scalar
    Random seed.

Returns
-------
y_trs : Vector
    Dependent variable for training set.
tx_trs : Matrix
    Explanatory variables for training set.
y_tes : Vector
    Dependent variable for testing set.
tx_tes : Matrix
    Explanatory variables for testing set.
```

Description of the procedure.

In this section, we will describe the steps we followed to clean the data, augment our matrix of input variables and select our hyperparameters through cross-validation.

Data preparation.

We start by importing the data, transforming the dependent variable by giving the value 1 to “s” and 0 to “b”. Then the values -999 are replaced by NaN. We then separate the matrix of explanatory variables by deleting the “Id” and “Prediction” columns. We then extract the vector of dependent variables by keeping only the “Prediction” column.

Once the data is cleaned and adequately partitioned, the missing values are replaced by the mean of the feature.

Feature generation.

In this section, we will describe the steps we followed to clean the data, increase our matrix of explanatory variables and select our hyperparameters through cross-validation.

The first augmentation is to add a dummy variable that indicates whether a data point is missing or not for each column.

Then we perform a polynomial augmentation. The polynomial augmentation of each feature vector x_n is done by adding a polynomial basis of degree M.

$$\theta(x) := [1, x_n, x_n^2, \ldots, x_n^M].$$

We carry out this transformation for each of the explanatory variables and use a polynomial basis of degree 3 in our study.

Finally, the feature matrix is standardized using the z-score method.

$$z = \frac{x - \mu}{\sigma},$$

Cross validation steps.

In order to determine the hyperparameters that we will use to generate our final predictions, we use K-fold cross-validation.

We start by determining a grid, for which we will try all possible combinations of hyper-parameters.

Then, for each possible combination, we will separate our data into K=4 equal parts, perform the aforementioned steps to process and augment the data, train the model on K-1 parts, and record the performance on the last one. This procedure is repeated K times, with the training and testing being done on a different subset of the data each time. The average performance is recorded for the K folds and the procedure is repeated for each possible combination of hyperparameters. At the end of the procedure, we select the combination that allows us to have the highest average accuracy.

This final set of parameters are then used to train the model on the entire training set. This model is used to generate the final predictions that were submitted to the AICrowd platform.