Read Me - Bike Sharing Demand Prediction (Q4)

General Instructions:

<u>Dependencies:</u>

1) Numpy
2) Math
3) Matplotlib
4) Seaborn
5) Warnings
6) ast
7) datetime
8) random

<u>Running code</u>:

1. Run in Terminal - To run the code, run the following command in the terminal
Code : python3 Assignment1_Q4.py

2. Run in Google Colab -(<u>Highly Recommended</u>)
Upload the **'Assignment-1_Q4.ipynb'** file in Google colab and run
3. **Run in Local Machine (Jupyter Notebook)**
It will show indentation error as code was done in Google Colab where indentation is of 2 spaces and in Local Machine Jupyter Notebook the indentation is of 4 spaces.So it is highly recommended to run the ipynb file in Google Colab.

<u>More Instructions</u>:

1. If data-files are in a local machine, change the file path accordingly. If running the notebook on Google Colab change the file path accordingly.
2. The Hyper-parameter tuning cells (named accordingly in the code for better understanding) takes very long time to run as it uses all given combinations of Hyper-parameters. If you don't want to run it then skip the cell in notebook or comment out the appropriate codes in python script. Also the best hyper-parameter settings are added in the notebook, use it directly to train the models.
3. While running any training, first rerun the cell named **Poisson Regression** containing all the required functions for regression.

<u>Dataset :</u>

We are provided hourly rental data spanning two years. For this competition, the training set comprises the first 19 days of each month, while the test set is the 20th to the end of the month. You must predict the total count of bikes rented during each hour covered by the test set, using only information available prior to the rental period.

Train Dataset : [train.csv]

Test Dataset: [test.csv]

Data Description :

datetime - hourly date + timestamp
season - 1 = spring, 2 = summer, 3 = fall, 4 = winter
holiday - whether the day is considered a holiday
working day - whether the day is neither a weekend or holiday
weather - 1. Clear, Few clouds, Partly cloudy, Partly cloudy
2. Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
3. Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain+Mild Clouds
4. Heavy Rain + Ice Pellets + Thunderstorm + Mist, Snow + Fog

temp - temperature in Celsius
atemp - "feels like" temperature in Celsius
humidity - relative humidity
windspeed - wind speed
casual - number of non-registered user rentals initiated
registered - number of registered user rentals initiated
count - number of total rentals (Target Variable)

Importing the datasets :

Below are the default paths of the train and test datafiles. Change the paths according to the file locations-
train_data_path='/content/gdrive/My Drive/train.csv'
test_data_path='/content/gdrive/My Drive/test.csv'
Google Colab :
Upload the Train and Test datasets in google drive and run the following code.

```
from google.colab import drive
drive.mount("/content/gdrive")
```

For running in Local Machine comment the above two lines of code and start with below after importing the required libraries.

```
df_train=pd.read_csv(train_data_path)
df_test=pd.read_csv(test_data_path)
```

Description of Functions:

1. modified_dataset(dataset):

Function to create new features and modify existing features

Args: dataset : dataframe (either train or test)

Returns: Modified dataframe

2. count_plots(df):

Function to plot the count of dataset

Args: df : dataframe

Plots: Count data for the six categorical features

3. count_mean(df,print_statistics=0):

Function to plot the mean of "count" features for each categorical features in the given dataframe

Args: df : dataframe

print_statistics=0(default; if it is 1 then it prints the statistics calculated)

Plots: Mean of the "count" feature for the categorical features in given dataset

Returns: Dictionary containing the mean count statistics of the features of data

4. count_median(df):

Function to plot the median of "count" features for each categorical features in the given dataframe

Args: df : dataframe

Plots: Median of the "count" feature for the categorical features in given dataset

5. aggregated_dataframe(df,col):

Function to return the hour aggregated data frame for the feature given

Args: df : dataframe

Col: feature in dataframe that is passed

Returns : Dataframe

6. count_against_features(df):

Function to visualize both the average mean count of features and count of features in given dataset

Args: df : dataframe

Plots : Visualization of count data with other features

7. categorical_count(df)

Function to plot the count of dataset for categories

Args: df : dataframe

Plots: Count data for the categorical features

8. visualize(df):

Function to visualize both the average mean and median count of features and count of features in given dataset

Args: df : dataframe

Plots : Visualization of all counts of data of each category and mean and median count against categories

9. Correlation_Plot(data):

Function to visualize the correlation between the features

Args : data : Dataframe

Plots : Correlation Plot to see the correlation among the features

10.drop_features(df,cols_list):

Function to drop the features in the list given and returns the modified dataframe

Args: df : dataframe (either train or test)

cols_list: List of columns to be dropped from the dataframe

Returns: Modified dataframe with given features dropped

11. max_norm(x):

Function to Normalize using maximum value

Args: x : list of values of the features

Returns: Normalized list of values of the features

12. normalize_train_test(df):

Function to pass the list of columns for normalization for the given dataset

Args: df : dataframe (either train or test)

Returns: Modified dataframe with given normalized features values

13. train_validate_split(train_data):

Function splits the given data such that 80% as train and 20% as validation data

Args: df : dataframe (input features data or output feature data)

Returns: train and validation dataframe

14. arrayToDataFrame(data):

Function to convert the input array data to dataframe

Args: data : output data of type int

Returns: output dataframe

15. Add_const_colum(df):

Function to add constant column of all values 1 to the dataframe

Args: data : dataframe

Returns: dataframe with constant column added in beginning

16. dataFrameToArrays(df):

Function to convert the dataframe to numpy arrays

Args: df : dataframe(either train or test or validate)

Returns: arrays

17. init_weights(n,mode):

Function to Initialize Weights

Args: n : dimension for Weights

mode: "Zero" = Zero Weights

"Random"= Random Weights

Returns: Initialized Weights Size = (dimension+1 x 1)

18. y_pred(x,w):

Function for Poisson Regression Hypothesis h(x) = exp(w.T*x)

Args: x : Input Feature Matrix or Design Matrix w : Weights

Returns: Prediction Y = h(x) = exp(w.T*x) with Size = (batch_size x 1)

19. gradient_w(x,y,w,mode,reg = 0):

Function for gradient calculation for Poisson Regression

Args: x : Input Feature Matrix or Design Matrix

y : Ture Response Variable

w : Weights

mode: 'No' : No Regularization

'L1' : L1 Regularization

'L2' : L2 Regularization

reg:Regularization Hyper-parameter (set 0 if mode = 'No')

Returns: Gradient of Loss with respect to W with Size = (n+1 x 1)

20. loss(x,y_true,w,mode,reg = 0):

Function to compute loss (Negative Log-likelihood) for Poisson Regression

Args: x: Input Matrix or Design Matrix

y_true: True Response Variable

w: Weights

mode: 'No' : No Regularization

'L1' : L1 Regularization

'L2' : L2 Regularization

reg:Regularization Hyper-parameter (set 0 if mode = 'No')

Returns: Negative Log Likelihood to Minimize

21. prediction_loss(y_pred,y_true):

Function to compute prediction loss (Negative Log-likelihood)

Args: y_pred: Predicted Response Variable

Y_true: True Response Variable

Return: (Negative Log-likelihood) Loss on Predicted Response Variable

22. RMSLE(y_pred,y_true):

Function to compute RMSLE (Root Mean Squared Logarithmic Error)

Args: y_pred: Predicted Response Variable

Y_true: True Response Variable

Return: RMSLE Loss on Predicted Response Variable

23. RMSE(y_pred,y_true):

Function to compute RMSE (Root Mean Squared Error)

Args: y_pred: Predicted Response Variable

Y_true: True Response Variable

Return: RMSE Loss on Predicted Response Variable

24. train(x_tr,y_tr,epochs,alpha,mode,weight_mode,reg = 0, verbose = True):

Function for training model

Args: x_tr: Input Matrix or Design Matrix (Training Data)

y_tr: True Response variable

epochs: Number of Epochs for Training (Iteration for Gradient Descent)

alpha: Learning Rate

mode: 'No' : No Regularization

'L1' : L1 Regularization

'L2' : L2 Regularization

weight_mode: 'Zero' : Zero Initialization of Weights

'Random' : Random Initialization of Weights

reg: Regularization Hyper-parameter (set 0 if mode = 'No')

verbose: 'True' for Printing Loss Value Per Epochs

'False' for suppressing Printing Loss Per Epoch

Returns: Trained Weights Size = (n+1 x 1)

25. predict(x,w):

Function to predict response variable

Args: x: Input Matrix

w: Trained Weight

Returns: Predicted Response Variable

26. predict_count(y_pred,max_count):

Function to predict the complete count

Args: y_pred: Predicted Normalized Response Values

max_count: Maximum Count Encountered in Training Data

Returns: Predicted Response Variable in Proper Form

Run the above functions **(Function No. 17-26)** again whenever you want to train.

27. HyperParameterTuning(mode,x_tr,y_tr,x_val,y_val):

Function to return the dataframe with hyperparameters and corresponding train and validation loss for each hyperparameter value

Args :x_tr: Input Matrix or Design Matrix (Training Data)
y_tr: True Response variable
x_val: Input Matrix or Design Matrix (Validation Data)
y_val: True Response variable of validation data
mode: 'No' : No Regularization
'L1' : L1 Regularization
'L2' : L2 Regularization

Return : Dataframe containing the results of hyperparameter tuning

Running the above functions and getting results takes too much time to get the results. The results of hyperparameter tuning (for both L1 and L2 regularization) are given as an excel file and it should be stored in a drive while running the code.

L1 Regularization results data : [L1_hyperparameter.xlsx]

L2 Regularization results data : [L2_hyperparameter.xlsx]

28. feature_importance(mode,w):

Function to display the important features

Args : mode : 'No' : No Regularization
'L1' : L1 Regularization
'L2' : L2 Regularization
w : weights of the mode mentioned

Plots : Features and their corresponding weights

Important Features :

1. atemp
2. humidity
3. Hour
4. year

Best Hyper-Parameter Summary:

Regularization Mode	Hyper-parameters	Values
Un-regularized	Weight Initialization	Zero Initialization
	Learning Rate	0.0001
	Epochs	200
	Regularization Constant	0.0001
L1 Regularized	Weight Initialization	Random Initialization
	Learning Rate	0.0001
	Epochs	700
	Regularization Constant	0.0001
L2 Regularized	Weight Initialization	Random Initialization
	Learning Rate	0.0001
	Epochs	700
	Regularization Constant	0.0001

Test Loss :

Mode	Dataset	Negative Log Likelihood	Normalized RMSLE	Normalized RMSE	RMSLE	RMSE
Un-regularized	Validation	0.5957	0.1433	0.1980	1.1182	201.2788
	Test	0.4781	0.1191	0.1564	1.2377	155.0540
L1 Regularized	Validation	0.5976	0.1438	0.1981	1.0958	201.4554
	Test	0.4781	0.1191	0.1564	1.2377	155.0540
L2 Regularized	Validation	0.5976	0.1438	0.1981	1.0958	201.4554
	Test	0.4763	0.1175	0.1551	1.1816	153.8378