University of Colorado Denver

CSCI 4930/5930 Machine Learning, Fall 2021

Instructor: Ashis Kumer Biswas, PhD

Assignment 2

Background

I am quoting from the WebMD article [1]:
"A new baby's gender, name, time of birth, and birth weight are nice information for a birth announcement, but birth weight is especially important for an obstetrician. A large size at delivery has long been associated with an increased risk of injuries to a newborn and its mom. So the better a doctor can predict birth weight, the easier the delivery may be."

Ultrasound is a popular way of doing it. But, aha! You are a Data Scientist (or a "going to be one"(!)). You can amaze people by predicting the birth weight way earlier than ultrasound, right? In this assignment, let's accomplish this.

Dataset Description

In the "dataset/" you will find two csv and a txt files:

- **baby-weights-dataset.csv**
  - It has 101400 rows (samples) with 37 columns (variables). Each sample represents a case of a new-born. Very last column is "BWEIGHT", that tells the true weight of the new-born (in lbs unit). Actually, this needs to be considered as the target variable here.
- **data-description.txt**
  - Here, you will find the names of the 37 variables used in the dataset above. And, the source of the dataset did not offer me description of every single of them. But, after studying about them, I could elaborate only few of them. Please pardon my laziness. Okay, this file contains few descriptions for the variables. All the rest are mostly talking about the Mother's medical history and all. No big deal, I guess, for you to work with these variables without knowing their meaning.
- **judge-without-labels.csv**
  - This is an interesting file. It contains new samples: additional 2000 rows with 36 columns (without the BWEIGHT target column). Once again, this should be part of the training, as there are no ground truth target labels, right? Once the training is complete with the dataset provided above, you may want to apply your prediction algorithm to predict BWEIGHT values of these 2000 samples. We (me and the TA) will evaluate that as well.

Project Skeleton Description

You are given the project notebook in the "Assignment-02.zip". It contains the "dataset/" directories which contains the aforementioned items. It also loads the two csv files into two pandas dataframe variables. The notebook roughly provides 17 placeholders for you to implement / complete the following 17 tasks:

Task ID	Task	Points possible
1	Separate the full_dataset into two parts: X and y, where X denotes the input matrix containing only the input (i.e., independent explanatory) variables, and y denotes the target variable containing only the target values for exactly the same number of samples in the given full_dataset.	5
2	• Given X representing the input matrix from the full_dataset, y being the target vector (the rightmost column of the full_dataset), obtained from Task 1: • randomly split the (X,y) dataset into 75% for training and 25% for testing using the library function from the library sklearn.model_selection . Please pass to the train_test_split function an additional argument random_state=45931. • Store the 4 splits as X_train, X_test, y_train, y_test respectively. • Save the ID column for X_train and X_test into ID_train and ID_test as list variable. • Now, drop the ID columns from both X_train and X_test	5
3	Compute mean, stdev, min, max, 25% percentile, median and 75% percentile of BWEIGHT target variable (i.e, the target y) in the training set (i.e., y_train), and print the computed values as a numpy array containing these 7 results (respectively).	5
4	Given the training dataset (X_train, y_train), save as X_train_ohe after replacing all the non-numeric variables (i.e., categorical variables) with numeric encoding. Please consider using the "One-hot encoding" scheme i.e., introducing dummy variables. A brief description of the scheme can be found in the DUMMY-variables.note.txt file • Use the same encoder to perform onehotencoding on the X_test dataset and save the result as X_test_ohe. • Print the column names of X_test_ohe. *Hint: This task was intentionally solved (mostly) except the last part. Cheers!*	5
5	• Given the X_train_ohe (Onehot encoded Pandas Dataframe from Task 4), check if there are missing values, and if yes, count how many, and impute the missing values with corresponding mean values. • Finally, print the counting result as a Pandas dataframe named "missing_counts" having 2 columns {variable_name,num_of_missing_values). Please make sure that the result lists all the input variables in the given dataset.	5

Task ID	Task	Points possible
	• Now, impute the missing values by mean of the respective variable and save the revised dataframe as X_train_ohe_imputed.	
6	• Given a X_train_ohe_imputed (Pandas dataframe from Task 5) where all the categorical variables are already replaced with numeric values, print a list of top 20 highly correlated variables with respect to the target variable, and save the result as a Pandas dataframe named top20_df with 2 columns {variable,corr_score}. • Here, the corr_score between a variable x and the target variable y needs to be computed using the Pearson Correlation Coefficient (PCC). Please note, PCC ranges between -1 to +1. PCC score 0 means no correlation, while value towards +1 and -1 represent positive and negative correlations respectively. For instance, PCC=0.8 and PCC=-0.8 tell similar strength positive and negative correlations between the two subject variables. • Please do not include BWEIGHT in the top20_df list of top 20 correlated variable list.	5
7	Given the X_train_ohe_imputed (as Pandas dataframe from task 5) and and top20_df (as Pandas Dataframe from Task 6) having 2 columns {variable_name,corr_score} similar to the one you computed in Task 6: • Please save as X_train_t20 keeping only the columns listed in the top20_df dataframe. • Repeat the process for X_test_ohe (obtained from task 4), and save it as X_test_t20.	5
8	• Apply min-max scaling on the training dataset (X_train_t20 obtained from Task 7). Save the result as X_train_scaled_mm. • Then scale the test dataset (X_test_t20 obtained from Task 7) based on the metrics you obtain when you scale the training dataset. Save the result as X_test_scaled_mm. • PLEASE DO NOT SCALE y_train and y_test.	5

Task ID	Task	Points possible
9	• Apply standardization (i.e., normalization) scaling on the training dataset (X_train_t20 obtained from Task 7). Save the result as X_train_scaled_std. • Then scale the test dataset (X_test_t20 obtained from Task 7) based on the metrics you obtain when you scale the training dataset. Save the result as X_test_scaled_std. • PLEASE DO NOT SCALE y_train and y_test.	5
10	Given the (X_train_scaled_std, y_train) pairs denoting input matrix and output vector respectively: complete the three function definitions and demonstrate the functionalities of each by calling them with appropriate arguments as instructed below: • **linear_regression_closed_form_training** : It fits a linear regression model using the closed-form solution to obtain the coefficients, beta's, as a numpy array of m+1 values (Please recall class lecture), where *m* is the number of variables kept in X_train (the first argument to the function). Please measure the cpu_time needed during the training step. cpu_time is not equal to the wall_time. So, use time.perf_counter() for an accurate measurement. Documentation on this function can be found here: https://docs.python.org/3/library/time.html . Finally, the function returns betas (i.e., the m+1 beta values) and the cpu_time. • **linear_regression_closed_form_predict**: It takes a list of m+1 beta values (i.e., betas returned from the corresponding training function, and X_test (containing test samples each having *m* input variables). Now, using the provided beta values, predict each of the test samples provided, and let's name your prediction "y_pred". Return y_pred from the function. • **RMSE**: It takes two lists: y_test, y_pred, where the first list represents ground truth (i.e., actual) target values for the given samples, and the second list represents a corresponding predicted values for exactly same number of samples in y_test. Compute and return the Root Mean Squared Error (RMSE) of the prediction. • PLEASE DO NOT USE ANY LIBRARY FUNCTION THAT DOES THE LINEAR REGRESSION. • Now, call linear_regression_closed_form_training() function providing X_train_scaled_std, y_train obtained from Task 9, and save the returned results as betas_closed_form,cpu_time_closed_form. • Print betas_closed_form, cpu_time_closed_form	8

Task ID	Task	Points possible
	• Call linear_regression_closed_form_predict() function providing betas_closed_form,X_test_scaled_std obtained in Task 9. Save the returned result as y_pred. • Call RMSE() function providing y_test and y_pred. Save returned result as rmse_closed_form. • Print rmse_closed_form.	
11	Given the (X_train_scaled_std, y_train) pairs denoting input matrix and output vector respectively: complete the three function definitions and demonstrate the functionalities of each by calling them with appropriate arguments as instructed below: • **linear_regression_gd_batch_training** : It fits a linear regression model using the batch gradient descent algorithm to obtain the coefficients, beta's, as a numpy array of m+1 values, where *m* is the number of variables kept in X_train (the first argument to the function). Please use the alpha (i.e, the learning rate) and nEpoch (number of epochs) parameters in your implementation of the gradient descent algorithm. Please measure the cpu_time needed during the training step. cpu_time is not equal to the wall_time. So, use time.perf_counter() for an accurate measurement. Documentation on this function can be found here: https://docs.python.org/3/library/time.html . Finally, the function returns betas (i.e., the m+1 beta values) and the cpu_time. • **linear_regression_gd_batch_predict**: It takes a list of m+1 beta values (i.e., betas returned from the corresponding training function, and X_test (containing test samples each having *m* input variables). Now, using the provided beta values, predict each of the test samples provided, and let's name your prediction "y_pred". Return y_pred from the function. • PLEASE DO NOT USE ANY LIBRARY FUNCTION THAT DOES THE LINEAR REGRESSION. • Now, call linear_regression_gd_batch_training() function providing X_train_scaled_std, y_train obtained from Task 9, and alpha=0.01,nEpoch=1000, and save the returned results as betas_batch,cpu_time_batch. • Print betas_batch, cpu_time_batch • Call linear_regression_gd_batch_predict() function providing betas_batch,X_test_scaled_std obtained in Task 9. Save the returned result as y_pred. • Call RMSE() function providing y_test and y_pred. Save returned result as rmse_batch. • Print rmse_batch.	8

Task ID	Task	Points possible
12	Given the (X_train_scaled_std, y_train) pairs denoting input matrix and output vector respectively: complete the three function definitions and demonstrate the functionalities of each by calling them with appropriate arguments as instructed below: • **linear_regression_gd_stochastic_training** : It fits a linear regression model using the stochastic gradient descent algorithm to obtain the coefficients, beta's, as a numpy array of m+1 values, where *m* is the number of variables kept in X_train (the first argument to the function). Please use the alpha (i.e, the learning rate), nEpoch (number of epochs), nIteration (number of iterations) parameters in your implementation of the gradient descent algorithm. Please measure the cpu_time needed during the training step. cpu_time is not equal to the wall_time. So, use time.perf_counter() for an accurate measurement. Documentation on this function can be found here: https://docs.python.org/3/library/time.html . Finally, the function returns betas (i.e., the m+1 beta values) and the cpu_time. • **linear_regression_gd_stochastic_predict**: It takes a list of m+1 beta values (i.e., betas returned from the corresponding training function, and X_test (containing test samples each having *m* input variables). Now, using the provided beta values, predict each of the test samples provided, and let's name your prediction "y_pred". Return y_pred from the function. • PLEASE DO NOT USE ANY LIBRARY FUNCTION THAT DOES THE LINEAR REGRESSION. • Now, call linear_regression_gd_stochastic_training() function providing X_train_scaled_std, y_train obtained from Task 9, and alpha=0.01,nEpoch=50, nIteration=100 , and save the returned results as betas_stochastic,cpu_time_stochastic. • Print betas_stochastic, cpu_time_stochastic • Call linear_regression_gd_stochastic_predict() function providing betas_stochastic,X_test_scaled_std obtained in Task 9. Save the returned result as y_pred. • Call RMSE() function providing y_test and y_pred. Save returned result as rmse_stochastic. • Print rmse_stochastic	8
13	Given the (X_train_scaled_std, y_train) pairs denoting input matrix and output vector respectively: complete the three function definitions and demonstrate the functionalities of each by calling them with appropriate arguments as instructed below:	8

Task ID	Task	Points possible
	• **linear_regression_gd_minibatch_training** : It fits a linear regression model using the minibatch gradient descent algorithm to obtain the coefficients, beta's, as a numpy array of m+1 values, where *m* is the number of variables kept in X_train (the first argument to the function). Please use the alpha (i.e, the learning rate), nEpoch (number of epochs), nIteration (number of iterations), and batch_size parameters in your implementation of the gradient descent algorithm. Please measure the cpu_time needed during the training step. cpu_time is not equal to the wall_time. So, use time.perf_counter() for an accurate measurement. Documentation on this function can be found here: https://docs.python.org/3/library/time.html . Finally, the function returns betas (i.e., the m+1 beta values) and the cpu_time. • **linear_regression_gd_minibatch_predict**: It takes a list of m+1 beta values (i.e., betas returned from the corresponding training function, and X_test (containing test samples each having *m* input variables). Now, using the provided beta values, predict each of the test samples provided, and let's name your prediction "y_pred". Return y_pred from the function. • PLEASE DO NOT USE ANY LIBRARY FUNCTION THAT DOES THE LINEAR REGRESSION. • Now, call linear_regression_gd_minibatch_training() function providing X_train_scaled_std, y_train obtained from Task 9, and alpha=0.001,nEpoch=50, nIteration=1000,batch_size=32, and save the returned results as betas_minibatch,cpu_time_minibatch. • Print betas_minibatch, cpu_time_minibatch • Call linear_regression_gd_minibatch_predict() function providing betas_minibatch,X_test_scaled_std obtained in Task 9. Save the returned result as y_pred. • Call RMSE() function providing y_test and y_pred. Save returned result as rmse_minibatch. • Print rmse_minibatch.	
14	Given the 4 sets of results from the 4 experiments (from Tasks 10, 11, 12, 13) with closed form solution, batch gradient descent, stochastic gradient descent and mini-batch gradient descent, print a string from the set {"closed-form", "batch-GD", "stochastic-GD", "minibatch-GD"} that demonstrated the best predictive performance in terms of RMSE. .	5

Task ID	Task	Points possible
15	Given the 4 sets of results from the 4 experiments (from Tasks 10, 11, 12, 13) with closed form solution, batch gradient descent, stochastic gradient descent and mini-batch gradient descent, print a string from the set {"closed-form", "batch-GD", "stochastic-GD", "minibatch-GD"} that demonstrated the least training cpu time.	5
16	Given the (X_train_scaled_std, y_train) pairs denoting input matrix and output vector respectively, • call your implementation of Task 12: stochastic gradient descent based linear regression for each of these learning rates: {0.0001, 0.001, 0.05, 0.01, 0.1, 1.0} ▪ Please use the nIteration (number of iterations), nEpoch (number of epoch) parameters in your implementation of the gradient descent algorithm. • For each of the linear regression model, using the computed beta values, predict the test samples provided in the "X_test_scaled_std" argument, and let's name your prediction "y_pred". • Compute Root Mean Squared Error (RMSE) of your prediction using the RMSE() function you defined in Task 10. • Finally, print the learning rate that shows the best test performance, and also print as a pandas dataframe named summary with 2 columns: {learning_rate, test_RMSE} containing RMSE's of the 6 linear regression models. Also, print the best performing learning rate. • PLEASE DO NOT USE ANY LIBRARY FUNCTION THAT DOES THE LINEAR REGRESSION.	8
17	• Utilizing the best trained linear regression model (so far), predict the target for each of the samples in the judge_dataset. ▪ I believe you will not forget to do the following before call in the prediction algorithm: o Save the ID values of the judge dataset into ID_judge and drop it from the judge dataframe. o Perform onehot encoding using the same encoder you used to encode X_test (Task 4). o keep only the same top 20 variables as you did in Task 7. o scale the input variables based on the same metrics you used to scale the training dataset (Task 9). ▪ Now, call the prediction function of that model to obtain y_pred.	5

Task ID	Task	Points possible
	▪ Prepare and print as a pandas dataframe having columns: {ID, BWEIGHT}, where ID will the ID of the judge sample, and BWEIGHT is the corresponding y_pred value from your model prediction. • PLEASE DO NOT USE ANY LIBRARY FUNCTION THAT DOES THE LINEAR REGRESSION.	
	Total	100

Graduate Students only

Task 18: (Points possible: 20)

Bring your best model and submit your solution at https://www.kaggle.com/c/birth-weight-prediction Submit multiple entries. Demonstrate your effort by pushing yourself to improve your own score or beat other submissions (if any available) until the deadline and document your scores and list what changes you made in your submitted solution.

Implementation Requirements

- Please make sure to create a python virtual environment set up using the specific packages (with the noted version) listed in the "requirements.txt" and work in it.
- Other than numpy, pandas and the ones already provided in the notebook, you cannot import any other libraries while solving this assignment.
- Feel free to add as many cells in the jupyter notebook provided. But PLEASE DO NOT DELETE ANY OF THE PROVIDED CELLS.

Submission

In canvas, submit the jupyter notebook file after saving all the outputs from executing the entire notebook and nothing else.

Grading Rubrics

- Five of the 17 tasks have weight 8 points, and each of the remaining twelve tasks worth 5 points each. ***And, for the graduate students: the total score is 120 points and will be scaled to 100 after grading is complete which will be reflected in the gradebook.***
- Hard-coded implementations will be penalized. Please show your work by coding the algorithm, rather than returning only the answer.
- This is an individual assignment. The MOSS system is going to be used in functional level to check for collusion and plagiarism. Any red flag by the system will be reported to the University, and you will fail this course immediately.
- PLEASE DO NOT USE ANY LIBRARY FUNCTION THAT DOES THE LINEAR REGRESSION