COL 774 - Machine Learning - Assignment 1

Due Date:
7:00 pm on Sunday, 18th August, 2024 - Part 1 (Linear Regression)
To Be Declared - Part 2 (Logistic Regression)
Max Marks : 100

Notes:

- **Copyright Claim:** This is a property of Prof. Rahul Garg Indian Institute of Technology Delhi, any replication of this without explicit permissions on any websites/sources will be considered violation to the copyright.
- All the announcements related to the assignment will be made on Piazza. The access code is wbd8avkblhb. You are strongly advised to have a look at the relevant Piazza post regularly.
- This assignment has two parts: 1.1 – Linear Regression and 1.2 – Logistic Regression. 1.2 would be released later.
- You are advised to use vector operations using numpy or scipy (wherever possible) for best performance.
- You should use Python 3 only for all your programming solutions.
- Your assignments will be auto-graded on our servers, make sure you test your programs with the provided evaluation scripts before submitting. We will use your code to train the model on a different training data set and predict on a different test set as well.
- Input/output format, submission format and other details are included. Your programs should be modular enough to accept specified parameters.
- You may submit the assignment **individually or in groups of 2**. No additional resource or library is allowed except for the ones specifically mentioned in the assignment statement. If you still wish to use some other library, please consult on piazza. If you choose to use any external resource or copy any part of this assignment, you will be awarded D or F grade or **DISCO**.
- *Graceful degradation:* For each late day from the deadline, there will be a penalty of 7%. So, if you submit, say, 4 days after the deadline, you will be graded out of 72% of the weightage of the assignment. Any submission made after 7 days would see a fixed penalty of 50%.
- For doubts, use Piazza. Send an email to Vipul Garg for doubts that may disclose implementation details. You are also heavily encouraged to report your best objective function score on the given test set for part C) through mailing.

History:

- 10/08/24: Linear Regression Problem Statement Released. Logistic Regression would be released in the coming week, with expected deadline as 25^{th}.
- 12/08/24: Expected value of best λ for ridge regression test case is 3.0.

1. **Linear Regression - (50 points)**

Release date: Aug. 10, 2024, Due date: Aug. 18, 2024)

In this problem, we will use Linear Regression to predict the Total Costs of a hospital patient. You have been provided with the training dataset (and evaluation scripts) of the **SPARCS Hospital dataset**. Train Data is given as train.csv with the target as the last value; please refer to the Evaluation Appendix for more details, submission format, and evaluation on the sample test set(given as test.csv, with the last value missing). The final training and evaluation will be done on an unseen dataset. Ensure your output adheres to the format given in the Appendix using the evaluation script.

On a side note, you should know that the dataset above is actually a processed subset of this original dataset, which you are encouraged to look at. The original dataset has been pre-processed by us to make it suitable for linear regression. For those interested, the details of pre-processing can be found in the Appendix at the end of the document.

Make sure your programs run efficiently and do not end up using too much memory or CPU. For parts (a) and (b), your program should run using less than 8GB of RAM within 15 minutes on an *Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz* (see this link for theoretical peak GFLOPS of this processor). If your program is running within 10 minutes on your systems, things should be fine. For parts (a) and (b), you MUST NOT shuffle the training data to avoid a mismatch with the model solution.

(a) **(12.5 points)** Given a training dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, recall that linear regression optimization problem can be written as:

$$\min_{w,b} \sum_{i=1}^{n} ||w^T x^{(i)} + b - y^{(i)}||^2 \tag{1}$$

Here, (w, b) represents the hyper-plane fitting the data. This is useful for cases where each training sample is equally important. But there might be scenarios where we would want the model to treat some examples as less important (for example, in the case of outliers). In such cases, we turn towards **Weighted Linear Regression**. Its optimization problem can be written as:

$$\min_{w,b} \sum_{i=1}^{n} ||u_i(w^T x^{(i)} + b - y^{(i)})||^2 \tag{2}$$

Here, u_i is the weight depicting the importance of the i^{th} sample. It is known beforehand for each sample. Note that in matrix form, this error term can be written as:

$$(Y - XW)^T U (Y - XW) \tag{3}$$

Here, U is a diagonal matrix with U_{ii} as the square of the weight of the i^{th} sample$(= u_i^2)$. In the case of simple linear regression, all the terms of this diagonal matrix are equal.
Implement weighted linear regression on this dataset using normal equations (analytic solution). Note that you would need to work around the fact that you cannot explicitly create the U matrix(think why!). The weights will be given in the form of txt files. Let there be n training samples. The weights file would a be txt file containing n lines, where the i^{th} line denotes the **square of the weight of the i^{th} sample**. Note that you may add a column with all one feature in the matrix x to absorb the parameter b in the weight vector w. You may not use any library besides NumPy or SciPy for this part. Pandas is also allowed.

(b) **(12.5 points)** Implement ridge regression using normal equations to find the optimal hyper-plane. The ridge regression formula is:

$$\min_{w,b} \frac{1}{2} \sum_{i=1}^{n} ||w^T x^{(i)} + b - y^{(i)}||^2 + \frac{\lambda}{2} \Big(\sum_{i=1}^{m} ||w_i||^2 + ||b||^2 \Big) \tag{4}$$

Here, λ is the regularization parameter. Again, for simplicity and elegance, you may add an all ones features to all the examples (x) such that the parameter b is absorbed in w which now becomes $m + 1$ dimensional vector. You should use 10 fold cross-validation to determine optimal value of regularization parameter. Perform the cross-validation on train.csv only. Take the sum of the mean

square errors found on the 10 validation sets to compute the cross-validation loss for a certain λ. More formally, define the cross-validation error as:

$$\sum_{i=1}^{10} MSE_{validationset_i} \tag{5}$$

$$MSE_{validationset_i} = \frac{\sum_{i=1}^{n}(y_{true} - y_{pred})^2}{n} \tag{6}$$

In the evaluation, λ values would be given as line-separated values in a text file. Refer to the evaluation appendix for more details. You should use modified Pseudo-Inverse expression, as discussed in the class, for solving this part. You may not use any library other than NumPy or SciPy for this part. Pandas is also allowed. Best λ for the given test case is 3.0

(c) **(25 points)** Feature creation and selection [1] [2] [3] are important part of machine learning. In fact, a large part of pre-neural network machine learning is comprised of engineering the right features for the problem at hand.

The objective of this part of the assignment is to get the best possible prediction on unseen data(see objective function later) by creating additional features. For this part you can use train.csv, as your training dataset. Extend your data by creating as many features as you like. However, make sure that the final number of features that your model uses is less than 300. This part is known as **Feature Selection**. Think about why having, let's say, 1000 features would be troublesome for the computation of the Moore-Penrose pseudo-inverse (*Hint: There is an $O(m^3)$ term in its time complexity*).

One method for feature selection is Lasso regression, which will automatically select a small number (less than 300) of features. You will need to select the optimal regularization penalty λ for lasso regression that will give the best performance on unseen data. Use cross-validation on the training data for choosing the best regularization parameter λ. You might want to iteratively reduce the weights of outlier examples before passing the data to lasso for feature selection. See Objective Function for Part (C) later.You should try out different transformations to get best performance. But using Lasso Regression is optional. You are free to explore more methods of feature engineering and feature selection.

Make a 1-2 page report.pdf file stating the number of features in your final selection and also a brief reason for creating those features and why they are relevant to the underlying dataset. Also, explain your feature selection strategy briefly. Submit this report on Gradescope while the code files need to be submitted on Moodle. **The grading for this part will be relative and depend very heavily on the accuracy of your predictions.**

You are encouraged to use 'Lasso model fit with Least Angle Regression (Lars)' package/function for this part in case you choose to use Lasso Regression. *Note*: LARS package is available for Python. Click here for more details. **But feel free to explore more methods to get the best results!** Along with Numpy and Scipy, you are **free to use the Sklearn Library for feature engineering, feature selection, and linear regression**.

Important: You can (and should) experiment with any number of features on training data. However, while submitting your work, the total number of features in your final selection must be **less than 300** as we are going to do the final evaluation on a bigger data-set. Your submission file *linear_competitive.py* should create your *selected features* and then run the linear regression model training on the training set using the above features, and then do prediction on the test set. Your program will be run on very large train and test data – so make sure that you write efficient codes and you don't consume too much memory. We will run a reference code to figure out the memory and running time requirements for the evaluation run and will make sure that your program is given sufficient memory and CPU time to run up to 300 features. Again, if your program runs in under **15 minutes** in your machines, things should be fine. However, if your program doesn't complete in the stipulated time and memory, it may cause an early termination and you may get a 0 for this part. Consult the TA in charge incase you think your submission may require more run time.

Your submission file *feature_selection.py* should contain all your code for feature creation and selection using Lasso or any other method. However, when running linear_competitive.py (to save runtime), you are not expected to run feature_selection.py. You may directly code your finally selected features in linear_competitive.py.

Some examples of feature generation are (i) One-Hot encoding (ii) using average/median values of the target for all the examples with a particular value of a categorical variable (e.g., *Facility_id* = 1453) (iii) average/median target value for all the examples with two specified categorical values (e.g., *Facility_id* = 1453 and *CCSR Diagnosis Code* = 13) and others examples discussed in the class. Application of common sense and innovative thinking about the domain will greatly help create meaningful features. **Note** that we have followed an ordinal encoding for categorical variables(see data pre-processing section towards the end of the pdf). A quick thought would reveal that this is not optimal for our use case; and can be improved!

OBJECTIVE FUNCTION FOR PART C

Instead of computing the normal root mean square error by incorporating all the samples in the test set, we want to do something about the outliers. We don't want their presence to massively influence the error of our model. In other words, we are happy to make really bad predictions on these outliers as long as we are making good predictions on the rest of the dataset. More concretely, we want to minimise the root mean square error of the best 90% predictions that our model makes on the test set. Mathematically, your model needs to minimise the following on the test set:

$$\sqrt{\frac{\sum_{i=1}^{0.9n} sortedE_i^2}{0.9n}} \tag{7}$$

Here, *sortedE* is a list obtained by sorting the errors obtained on each sample by the model on the test set in ascending order, where the error is defined as $||predicted(Y) - Actual(Y)||$.

You may try different loss functions than the standard MSE loss to optimize the objective function. One approach might be related to weighted linear regression done in part (a). Another approach may be to optimize L1 loss instead of L2 loss to make it less sensitive to outliers. Please note the difference between the loss function(the function that the model tries to optimize on the training set while training) and the objective function(the function that computes the performance of the model on the test set). The loss functions generally need to be less complex so that the model can optimise them using some mathematics, while the objective functions can be pretty much anything as long as it makes sense.

What have you been given?

- A training set (train.csv) and a test set (test.csv, without the total costs column). You have also been given a test_pred.csv file that contains the total costs column of the test.csv file. You can use this to test your model performance on the test set.
- sample_weights1.txt and sample_weights2.txt, which contain two sets of weights for the samples for task (a). These are just 2 test cases. A single sample_weight file would act as a command line argument for your program. You have also been given expected_weightsa1.txt and expected_weightsa2.txt, which contain the expected model parameters(bias in the first line and the rest of the parameters after) when it is trained using sample_weights1.txt and sample_weights2.txt, respectively. Note that since the pseudo inverse is deterministic, your model parameters should match the expected weights. You have also been given preda1.txt and preda2.txt, which contain the predictions on the test set that the respective expected model is supposed to make. Again, your predictions on the test set for part (a) must match the predictions present in these files.
- regularization.txt, which contains the list of lambda values for task (b). This would act as a command-line argument for your program. Again, you have been given expected_weightsb.txt and predb.txt files that serve the same purpose as above. bestlambda.txt contains the best value of λ for the given test case. Ensure correctness by comparing your best λ with this.
- grade_a.py, grade_b.py, and grade_c.py that act as evaluation scripts for the respective parts. Note that we will be using different training and test sets during the evaluation.

Submission Instructions:

- You must submit 3 files: linear.py(contains code for part a and b), linear_competitive.py(contains code for part c) and feature_selection.py(contains code for feature creation and selection for part c).
- For part (a), linear.py would be called as follows:
  python3 linear.py a train.csv test.csv sampleweights.txt modelpredictions.txt modelweights.txt
  Here, you have to write the predictions (1 per line) on the test set in the modelpredictions.txt. Also, output your model weights in modelweights.txt (intercept in the very first line).
- For part (b), linear.py would be called as follows:
  python3 linear.py b train.csv test.csv regularization.txt modelpredictions.txt modelweights.txt bestlambda.txt
  Here, regularization.txt is a list of line-separated λ values. You must perform 10-fold cross-validation for all the λ values and report the predictions and weights as in part (a). Additionally, you must output the best regularization parameter in bestlambda.txt.
- For part c), linear_competitive.py would be used. It will be called as follows:
  python3 linear_competitive.py train.csv test.csv output.txt
  Your file must create and use your best features(less than 300) to train on train.csv and calculate predictions on test.csv. These predictions must be output in the output.txt in the same order as the test.csv samples. In addition, you will also submit feature_selection.py that contains code of you feature engineering and feature selection process. The selected features in this file must match with the features you end up using in linear_competitive.py. We should be able to run this file with the following command:
  python3 feature_selection.py train.csv created.txt selected.txt
  Here, your code will create 2 files:
  i. created.txt : It will contain the expressions of the features that you created(one per line). For example, if the features that you created are x_1, x_2, x_1x_2,Male, Female,Other (The last 3 are one-hot encodings of gender), your created.txt may look like:
  x_1
  x_2
  x_1x_2
  Male
  Female
  Other
  ii. selected.txt: It will contain binary values indicating the features that were selected out of all the features created in created.txt. The i^{th} line of the file would be a 0/1 indicating whether feature i of created.txt was selected or not. For example, if you choose x_1, x_1x_2,Male as your final features, your selected.txt will look like:

1
0
1
1
0
0

- All the 3 files must be put in a folder. The name of the folder would be Entrynum for individual submissions and Entrynum1_Entrynum2 for group submissions. For example, if 2020CS50450 plans to do the assignment alone, he must name this folder 2020CS50450. If he plans to submit it in collaboration with 2020CS50449, the folder name must be 2020CS50450_2020CS50449 or 2020CS50449_2020CS50450.
- This folder must then be zipped, and the name of the zip must be foldername.zip. This zip file must be uploaded on moodle by any one of the group members.
- Additionally, your report.pdf for part c) must be submitted on Gradescope. Kindly ensure to add your partner in case you are submitting in groups.
- Don't submit any dataset files!

Evaluation:

- **Training:** You are given file train.csv and test.csv. you can get 0 (in case your program gives an error), partial marks (if your code runs fine but predictions are incorrect within some predefined threshold) and full (if your code works exactly as expected). The final scores for these parts will be calculated using unseen data. Make sure you name your files exactly as given in the command line arguments. You can check your output format by using the evaluation scripts.
- For part (a) : python3 grade_a.py outputfile_a.txt weightfile_a.txt preda1.txt expected_weightsa1.txt where, the outputfile_a.txt and weightfile_a.txt are created by running your linear.py with argument 'a'. Note that you can also use files associated with the 2nd test case as well.
- For part (b) : python3 grade_b.py outputfile_b.txt weightfile_b.txt predb.txt expected_weightsb.txt where the outputfile_b.txt and weightfile_b.txt are created by running your linear.py with argument 'b'.
- For part (c), marks will be based on the objective function on an unseen test set. There will be relative grading for this part. You can see the performance of your predictions on the given test set by running the following: python3 grade_c.py outfile.txt test_pred.csv, where outfile.txt is the file produced by running your linear_competitive.py.
- Please refer to the submission instructions, any submissions not following the guidelines will fail the auto-grading and be awarded 0. There will be **NO DEMO** for this assignment.

Extra Readings (highly recommended for part (c)):

(a) Regression Shrinkage and Selection Via the Lasso

(b) Compressive Sensing Resources

(c) Least Angle Regression

Data Pre-processing description

Any real dataset will rarely be ideal. In this dataset, we have done the following pre-processing (in order):

- Removed rows(a small fraction) which had one or more values missing.
- Removed the length of stay and the total charges(different from total cost) column from the dataset.
- Ordinal encoding of all columns which contained categorical data with the mapping present in mapping.txt(present in Google Drive Linked folder).

Note: This section has only been provided to demonstrate the extra efforts that go in data preparation before applying a model.