COL 774 - Machine Learning - Assignment 1

Due Date:
7:00 pm on Sunday, 18th August, 2024 - Part 1 (Linear Regression)
7:00 pm on Sunday, 26th August, 2024 - 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.
- 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~~.
- 14/08/24: Part-1(Linear Regression): Additional implementation details added for the task (a) and (b). Some pointers for task (c). Please download latest zip file from the drive link. Gradescope link is also up!
- 18/08/24: Part-2 (Logistic Regression) released!
- 21/08/24: Part-2 (Logistic Regression) part (a) test cases have been released. Evaluation scripts would released tomorrow.

- 22/08/24: Part-2 (Logistic Regression) Added a missing minus sign in cost function of part (b).
- 22/08/24: Part-2 (Logistic Regression) deadline has been extended by 1 day. It's now 7:00 PM, 26/08/24.
- 22/08/24: Part-2 (Logistic Regression) Evaluation scripts have now been added!
- 23/08/24: Part-2 (Logistic Regression) Additional leads have been added for part (b)

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. Please use numpy.linalg.inv to compute the inverses. The matrix that you would need to invert would be invertible. You don't need to worry about that.

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

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

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. To make the sizes of each validation set uniform, don't consider the final 4 rows of train.csv for this part. You can do so by doing X_{new} = X[:-4] and Y_{new} = Y[:-4] where X and Y are numpy arrays containing the entire train.csv. Even in the final evaluation, you can assume that your code should ignore the final 4 rows of the given training file. Take the first $N/10$ rows in the first validation set, the second $N/10$ rows in the second validation set, and so on. 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:

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

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

Here, n is the number of samples in the validation set(=N/10).
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 a modified Pseudo-Inverse expression$((X^TX+\lambda I)^{-1}X^TY)$, as discussed in the class, for solving this part.Please use numpy.linalg.inv to compute the inverse, and not numpy.linalg.pinv to maintain consistency. The difference is due to different floating point errors that happen in both of these algorithms. You may read about the differences. 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~~. Please use Numpy/SciPy only to split the training set, not Scikit. Also check the *what have you been given?* section for more information.

(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. You may use PCA for dimensionality reduction as well. It's an algorithm that reduces the dimensionality of data while trying to lose as little information as possible. You may read about it. Please check the updated submission instructions.
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**. Please note that while any Sklearn algorithm is allowed for feature engineering, you may only use LINEAR REGRESSION as your machine-learning model for making predictions. You may add any regularization or make changes to the loss term.

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). Note that this file has been changed. Please use the new file in the Google Drive Zip. 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. Since the regularization.txt has changed, bestlambda.txt has also changed. The new best value of λ is 5.0. You have additionally been given a crossvalidation_errors.txt. It contains pairs of λ and cross-validation errors obtained. This file is not part of the evaluation. You can just use this to check whether you are getting the right cross-validation errors.
- grade_a.py, grade_b.py, and grade_c.py that act as evaluation scripts for the respective parts. There has been a change in the grade_b.py script. It now also checks the computed λ for correctness. 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 the code of your 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. In PCA, every principle component is just a linear combination of the original features. Incase you wish to use PCA, you should be able to interpret the engineered features as linear combination of the original features. For example, if the first principle component is $0.4x_1 + 0.6x_2$ and you wish to use it, the same must be output by your feature_selection.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
$x_1 x_2$
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~~ python3 grade_b.py outputfile_b.txt weightfile_b.txt predictedlambda.txt predb.txt expected_weightsb.txt bestlambda.txt
  where the outputfile_b.txt, weightfile_b.txt and predictedlambda.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
(d) PCA

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 into data preparation before applying a model.

2. **Logistic Regression (50 points)**

   Release date: 18/08/24, Due date: 26/08/24

   In this problem, we will use Logistic Regression to build a classifier for **Hospital Inpatient Discharges** training data. In parts (a) and (b), you will be building a logistic regression model for 4 class classifications to predict the 'Race' (target column in the given dataset) of the patients in the Hospital. The 'Race' can be 'Black/African American'(class 1), 'Multi-racial'(class 2), 'Other'(class 3), or 'White'(class 4). For part (c), you will build a logistic regression model for binary classification to predict the 'Gender' : 'F'(class -1) and 'M'(class 1).

   For part (a) and (b) train1.csv, test1.csv and testpred1.csv(containing target values of test1.csv) are the relevant files. Note that for these parts, you have been given a one hot encoded dataset. You have also been given a script **encode.py** that was used to create this one-hot encoded dataset from the original dataset.

   For part (c) train2.csv, test2.csv and testpred2.csv(containing target values of test2.csv) are the relevant files. These datasets contain the same **ordinal encoding** that was used in Linear Regression's mapping.txt.

   One-hot encoding is done in **encode.py** as follows: Suppose a column C has values from the set {234, 453, 121, 347} then the corresponding 4 columns of the one hot encoded columns of C be encoded in ascending order i.e., the ascending order of the set of values is {121, 234, 347, 453} so 121 be encoded as (1, 0, 0, 0), 234 as (0, 1, 0, 0), 347 as (0, 0, 1, 0), 453 as (0, 0, 0, 1). Next, we drop the first column in the one hot encoded columns of C to avoid the Dummy Variable Trap (For more information click here.) So now, 121 be encoded as (0, 0, 0), 234 as (1, 0, 0), 347 as (0, 1, 0), 453 as (0, 0, 1). You may have a look at encode.py for more details. Note that in this script, the set of values for a particular column hasn't been created using the train.csv itself because there may be some values of categorical variables that don't occur in the training set but occur in the test set. Instead, a file similar to mapping.txt(mapping.json) has been used.

   (a) **(25 points)** Append a column of ones as a first column in your data to absorb the bias terms in a single weight matrix. Now, your weight matrix will have shape ([n + 1] x 4) where n is the total number of columns(excluding those of target) after one hot encoding and the first row of the weight matrix will represent the bias terms. The first column of weights correspond to class1, the second column to class2 and so on.
   Do not shuffle rows or columns of the data within your code, otherwise your solution will not match the reference solution. The input data has already been shuffled. **Also, initialize all your weights with 0 for this part, so that your algorithm becomes deterministic and your solution can be compared with the model solution. Also, use float64 as your datatype.** You are not allowed to use any library other than NumPy and SciPy for this part apart from using pandas for data read.
   Given a training dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, recall that the log-likelihood for logistic regression (k-class) can be written as:

$$L(w) = \frac{-1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{k} \{y^{(i)} = j\} \log(g_{w_j}(x^{(i)})) \tag{8}$$

$$g_{w_j}(x) = \frac{e^{(w_j^T x)}}{\sum_{i=1}^{k} e^{(w_i^T x)}} \tag{9}$$

   (w) represents the decision surface learned by logistic regression and the 1st column in X is all ones to take care of constant bias term. $\{y^{(i)} = j\}$ is 1 when $y^{(i)}$ is equal to j and zero otherwise
   Instead of using the above loss function, we want to use a loss function that helps us in tackling the **class imbalance**. For this, we use the following loss function:

$$L(w) = \frac{-1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{k} \frac{\{y^{(i)} = j\} \log(g_{w_j}(x^{(i)}))}{freq_j} \tag{10}$$

$$g_{w_j}(x) = \frac{e^{(w_j^T x)}}{\sum_{i=1}^{k} e^{(w_i^T x)}} \tag{11}$$

Note the negative sign here(which means you need to minimise this function). Here, $freq_j$ is the frequency of class j in the **entire** training set(and not a particular batch). Implement a gradient **descent** algorithm and solve the weighted logistic regression problem using it.
Mini-batches and learning rate are critical elements in gradient descent-based algorithms.
You must implement mini-batch gradient to solve the above problem. Use batch size as user input. Run your code for a defined number of epochs. (Note that an epoch here means updating gradients with all samples or full data considered once, don't shuffle the data and start picking up batches sequentially from beginning of data matrix, i.e., in a single epoch, first update the gradients using x_index(0 to batchsize-1) then x_index(batchsize to batchsize*2-1) then x_index(batchsize*2 to batchsize*3-1) and so on. Also it maybe possible that the last batch size is smaller than the original batch size if sample size is not a multiple of batch size, in that case, do gradient updations according to batch size of the smaller batch). For every batch, use $freq_j$ as the frequency of class j in the **entire training dataset**. N would be batch size. **VERY IMPORTANT: KINDLY ENSURE THAT YOU ARE FOLLOWING THE EQUATION OF LOSS(AND ALSO THE GRADIENT OF THE LOSS) PROPERLY**
You must also experiment with following learning rate strategies:

i. Constant learning rate

ii. Adaptive learning rate $\eta_t = \frac{\eta_0}{1+kt}$ Here, t stands for the epoch number, and η_0, k are fixed hyperparameters. For example, if you consider batch number 5 in epoch number 2, t = 2 for that batch. If you consider batch number 44 in epoch number 1, t = 1 for that batch.

iii. Exact line search using ternary search.
Read about line search: Line Search
Let's say your current weights are w, and the gradient of the loss function at w is g. In gradient descent, you want to make a weight updated according to $w = w - \eta g$, where η is a suitable learning rate. In exact line search, you aim to find the η that minimizes $L(w - \eta g)$. Note that this is a one-dimensional function in η, and since it can be proved that multiclass logistic regression loss function is convex, 1-D convex optimisations technique can be applied on it to find such η. One such technique is ternary search. Read about it : Ternary Search.
In our context, to find the suitable η, following the below algorithm. **VERY IMPORTANT: The given algorithm has a slight bug. It is upon you to fix it! You may email me to confirm your corrections. But don't use piazza for this!**

Algorithm 1 Ternary Search

```
1: $\eta_l \leftarrow 0$                                        ▷ $\eta_0$ is hyper param
2: $\eta_h \leftarrow \eta_0$
3: $g = \nabla L(w)$
4: while $L(w) > L(w - \eta_h g)$ do
5:     $\eta_h = 2\eta_h$
6: end while                                 ▷ To find the boundaries of ternary search
7: for 20 iterations do
8:     $\eta_1 = \frac{2\eta_l + \eta_h}{3}$
9:     $\eta_2 = \frac{\eta_l + 2\eta_h}{3}$
10:    if $L(w - \eta_1 g) > L(w - \eta_2 g)$ then
11:        $\eta_h = \eta_2$
12:    else if $L(w - \eta_1 g) < L(w - \eta_2 g)$ then
13:        $\eta_l = \eta_1$
14:    else
15:        $\eta_l = \eta_1$ and $\eta_h = \eta_2$
16:    end if
17: end for
18: $\eta = \frac{\eta_l + \eta_h}{2}$                           ▷ To get final $\eta$.
```

After finding a suitable η, you can update your weights and repeat.

Your code should output the weights and run for the exact number of epochs provided. Refer to the submission instructions for more details.

(b) **(12.5 points)** Find the best algorithm and working hyper-parameters that minimize the loss as quickly as possible. and use the model to train and make predictions on the evaluation data. This part will have a fixed time (10 minutes) to train your model and predict on the test set. (Hint: Plot the loss function, L(w,b) with respect to number of floating point operations, and runtime varying i) Batch size, ii) Variant of the gradient descent algorithm and iii) Value of learning rate parameters (η_0, α, β for adaptive case), to visualize and select the best algorithm and working hyper-parameters). You may only vary hyperparameters in the algorithm that you implemented above. **You may change the initialization of the weights as well**. **You may also explore other learning rate strategies. But kindly note that all code must be yours.** ~~**No library apart from Numpy and Scipy will be allowed.**~~ You may use sklearn for the preprocessing described below. But please don't use it for anything else.

Please note that this is a competitive part, and your code would not be allowed to run for more than 10 minutes. Please use a time measuring library to keep track of time, and make predictions on the given test.csv timely. In case your code is not able to produce a prediction file, you will be given 0 for this part. Refer to the submission instructions for more details.
In this part, for every sample in the test.csv, your code will output a 4-tuple(softmax applied on the logits). This is a probability vector, where p_i is the probability of a sample belonging to i^{th} class. Your code must aim to minimize the following function on the test set(Note that minus sign was missing here earlier):

$$-\frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{k}\frac{\{y^{(i)}=j\}\log(p_j)}{freq_j} \tag{12}$$

Here, $freq_j$ is the frequency of j^{th} class in the test set. You can expect similar distributions of the frequencies as the training set because they have been sampled from the same original dataset.
After completing part (a) and running the given testcases, you may have been seen that while the loss is decreasing, the learning rate is extremely low, and the convergence is extremely slow. This can be partly attributed to the extreme sparsity of the dataset. Gradients become highly variable across different features, leading to poor convergence when applied by a common learning rate.
Optimizing the given loss function in such a case would become challenging, and some might say, impossible. To tackle this, we are allowing you guys to do feature scaling. You may use the following lines of code:
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_train = np.insert(X_train,0,np.ones(X_train.shape[0]),axis=1) (Inserting the columns of 1 for bias).
This is a simple preprocessing function. For each column of X_train, it normalises the values through the following formula:

$$x = \frac{x-\mu_{column}}{\sigma_{column}} \tag{13}$$

This normalizes the scale of each feature, effectively making the gradient descent smoother. You are encouraged to read more about feature scaling online.
IMPORTANT!: You need to apply the same preprocessing to the test set before making predictions. Note that the means and standard deviations that you use here must be those obtained from the TRAINING SET(think why!). You can apply the same transform on the test set as follows:
X_test = scaler.transform(X_test)
X_test = np.insert(X_test,0,np.ones(X_test.shape[0]),axis=1) (Inserting the columns of 1 for bias)
To create a common reference of what all you can do in part (b):

i. Apply the pre-processing given above. You may not apply any other pre-processing(you are free to do so in part (c)). Applying the pre-processing is optional, but recommended.
ii. Vary weight initialization. While it has extremely drastic effects in neural networks, it may not have as much effect in simple logistic regression. Still, you are encouraged to read about it.
iii. Vary learning rate strategies. You can explore new learning rate strategies as well. But all your code must be your own.
iv. Vary hyperparameters

(c) **(12.5 points)** In this task, you will build a model that performs binary classification. This part is competitive. The accuracy on the hidden test set will used as performance metric to rank all the submissions.

Feature engineering is an important aspect of Machine Learning. Create additional features that you think will be most predictive of the given target. Use a suitable feature selection algorithm along with cross-validation to select the final features that will be most predictive on the test set. Some examples of feature selection algorithms are Large-scale sparse logistic regression, Sparse multinomial logistic regression: fast algorithms and generalization bounds, ANOVA based feature selection.

For this part, you are provided data(train2.csv, test2.csv and test_pred2.csv) in a similar format as the linear regression tasks. You may adjust the encode.py to create one-hot encoding on this dataset, or you may use any other feature encoding method. You must train a model to do **Binary Classification** on this dataset. You must predict the Gender of the given samples: 'F'(Class:-1) and 'M'(Class: 1)

For this part, you may use any publicly available method or library for feature creation or selection. Your final number of features must be less than 1000.

With this new data, perform part (b) of the assignment for **binary classification**(you may use only one set of weights and do a sigmoid instead of softmax) and ensure that the entire process of data reading, training and testing takes less than 15 minutes. It is strongly advised to use a time measuring library to keep track of time, and create your predictions timely.

Kindly note that for training your model, you may ONLY use code from part(b) adapted for binary classification. You cannot use any other algorithm or library. For this part, you may use the category_encoders library as well to perform encoding. If you wish to use anything else, please consult on piazza or through mail.

Your submission file *feature_selection.py* should contain all your code for feature creation and selection. However, when running logistic_competitive.py (to save runtime), you are not expected to run feature_selection.py. You may directly code your finally selected features in logistic_competitive.py. Your logistic_competitive.py must output predictions(-1 or 1) on the given test dataset.

OBJECTIVE FUNCTION FOR PART C

Your submission would be judged based on the prediction accuracy on test data. There will be relative marking for this part. You may use a suitable loss function(by adjusting/removing sample weights, adding regularization, etc.). One suggestion for the loss function is $\sum_1^m \hat{y}_i y_i$, where $y_i = 2\sigma(g(w^t x)) - 1$

Make a detailed report stating your parameter tuning method, feature creation, selection and other experiments you performed to arrive at your final solution to parts (b) and (c). Submit this report on gradescope while the code files need to be submitted on moodle.

What have you been given?

- mapping.txt that contains the ordinal encoding applied on the raw dataset.
- mapping.json that contains the list of possible values(of ordinal encoding) for all the categorical variables.
- A training set (train1.csv) and a test set (test1.csv, without the Ethnicity column) for tasks (a) and (b). You have also been given a test_pred1.csv file that contains the Ethnicity column of the test1.csv file. You can use this to test your model performance on the test set.
- A training set (train2.csv) and a test set (test2.csv, without the Gender column) for the task (c). You have also been given a test_pred2.csv file that contains the Gender column of the test2.csv file. You can use this to test your model performance on the test set.
- **encode.py** that is the Python script used to create one-hot encodings. You may look at it and use it for the competitive part in your submission. It is an extremely naive implementation, and you may want to better it if you want to integrate it in your code.
  ~~**The files mentioned below will be added to the zip in some time. Meanwhile, you are advised to begin with your implementation and ensure that the training loss is decreasing.**~~
  Folder named tests has been added in the zip. I have additionally included a loss.txt for each test case. It's also for debugging purposes. Its output format must be self-explanatory. The focus of these testcases is to ensure that your loss continuously decreases. For the competitive part, you are encouraged to try out new weight initialization and learning rate variation techniques.
- A folder named tests. It contains test cases for part (a). Each test case is another folder containing 7 files: params.txt, finalweights.txt, weights1.txt, weights2.txt,....weights5.txt.
  (a) params.txt contains the input parameters to your gradient descent algorithm. It contains 4 lines. The 1st line is a number [1-3] indicating the learning rate strategy. The 2nd line is fixed learning rate(for (i)), comma separated η_0 and k (for (ii)), and η_0 (for (iii)). The 3rd line is the number of epochs. The 4th line is the batch size.
  (b) finalweights.txt contains the expected final weights of your algorithm. You may use this to check the correctness of your implementation. Check evaluation section for more details.
  (c) weights1.txt...weights5.txt contain the model weights after epoch 1, epoch 2..., and epoch 5, respectively. They are not part of the evaluation. You may use them to debug your implementations.

  Evaluation scripts have now been added.
- 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: logistic.py(contains code for part a and b), logistic_competitive.py(contains code for part c) and feature_selection.py(contains code for feature creation and selection for part c).
- For part (a), logistic.py would be called as follows:
  ~~python3 logistic.py a train1.csv sampleweights.txt modelweights.txt~~
  python3 logistic.py a train1.csv params.txt modelweights.txt
  Here, you need to output your model weights to modelweights.txt. params.txt is an input file that tells you about the gradient-descent parameters(as described in the above section). Write your weight matrix (which includes bias terms in the first row) by flattening the weight matrix row-wise, i.e., write the first row first (1 value per line), then the second row, and so on. Use the evaluation scripts to check your output format.
- For part (b), logistic.py would be called as follows:
  python3 logistic.py b train1.csv test.csv modelweights.txt modelpredictions.csv
  Here, output your final weights in modelweights.txt as described above. Also, create a modelpredictions.csv file that contains number of lines equal to the samples in the test set. Don't create a header. For every line, there are 4 comma-separated values: p1,p2,p3, and p4, which correspond to the probabilities of the sample being in class 1, 2,3, and 4, respectively.
- For part c), logistic_competitive.py would be used. It will be called as follows:
  python3 logistic_competitive.py train2.csv test.csv output.txt
  Kindly note the difference in the format of the data in train1.csv and train2.csv. The

same will be followed in our evaluation training and test sets. Your file must create and use your best features(less than 1000) to train on train.csv and calculate class predictions(-1 for female, 1 for male) 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 the code of your feature engineering and feature selection process. The selected features in this file must match with the features you end up using in logistic_competitive.py. In PCA, every principle component is just a linear combination of the original features. Incase you wish to use PCA, you should be able to interpret the engineered features as linear combination of the original features. For example, if the first principle component is $0.4x_1 + 0.6x_2$ and you wish to use it, the same must be output by your feature_selection.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:

(a) 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,Black, White, Other (The last 3 are one-hot encodings of Race), your created.txt may look like:
x_1
x_2
x_1x_2
Male
Female
Other

(b) 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,Black 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 (b) and (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:

- The final scores for all 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. **Note that the final training and test sets used in the evaluation would have similar sizes to the datasets given.** If the memory and time requirements are getting satisfied on the given sets, you can assume that they would work on the evaluation sets as well.
- For part (a): python3 grade_a.py modelweights.txt expectedweights.txt
where the modelweights.txt is created by running your logistic.py with argument 'a'. expectedweights.txt is the path of the expected weights file. **While testing, please ensure that you are using params.txt and expectedweights.txt from the same testcase!**.
- For part (b) : Marks will be based on the loss 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_b.py modelpredictions.csv test_pred1.csv
where the modelpredictions.csv is created by running your logistic.py with argument 'b'.

- For part (c), Marks will be based on the binary prediction accuracy 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 output.txt is the file produced by running your logistic_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.

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 total charges(different from total cost) column from the dataset.
- One-hot encoding of all the columns present in mapping.json(present in Google Drive Linked folder) to create train1.csv and test1.csv. You may refer to encode.py for more details.
- Ordinal encoding of all columns which contained categorical data with the mapping present in mapping.txt(present in Google Drive Linked folder) for test2.csv and train2.csv.

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