# Linear Models for Regression and Classification

**ELL409**: Assignment-1

Last updated: 24-08-2022 10:00 AM
Deadline to submit assignment: **13-09-2022 11:59 AM**

## Instructions

1. While you are free to discuss all aspects of the assignment with your classmates or others, your code and your results and report must be entirely your own. We will be checking for any copying (on MOSS), and this will be treated as plagiarism and dealt with accordingly. In case of any doubts in this regard, please ask the TAs.

2. To assist with the automatically graded portion of the evaluation, please use only Python to write your code. In case of any particular difficulty with using Python, please contact your TA or instructor and we can figure something out.

3. Note that your submission will be evaluated using held-out training and validation sets. The format of these will be the same as in your given data sets (see below).

4. In addition to the code, you should prepare a report, compiling all your results and your interpretation of them, along with your overall conclusions. In particular, you should attempt to answer all of the questions posed in the respective parts of the assignment below. Any graphs or other visualisations should also be included therein.

5. The schedule for demos/vivas will be announced by your respective TAs, in advance. If for any reason you cannot attend in your scheduled slot, you must arrange for an alternative slot with your TA well in advance. Last-minute requests for rescheduling will normally not be accepted.

6. The results must be reproducible. Failure to reproduce the results during demo(s) will result in penalty.

7. The submission instructions are appended at the end. Failure in submitting exactly using the guidelines will result in penalty.

8. This assignment will be semi-autograded. A component of the points will be based on the output prediction scores. Follow the submission instructions (below).

## Part 1: Multivariate Linear Regression (10 points)

The objective of this part of the assignment is to implement the concepts of linear regression that you have learnt in the class. In this part, you will be implementing a multivariate linear regression, where the task is to learn a function that maps an input variable $x_i$ to a continuous target variable $t$ across $N$ features (as shown in Eq. 1).

$$t = w_0 + \sum_{i=1}^{N} w_i x_i + \epsilon \tag{1}$$

The relation (shown in Eq. 1) has certain noise $\epsilon \sim \mathcal{N}(0, \Sigma)$ that is fixed for the whole dataset (i.e. it is drawn from this distribution for every datapoint). Each sample from the dataset has six features $(x_1, x_2, x_3, x_4, x_5, x_6)$ and a target variable $t$. Implement a multivariate regression model to achieve the following sub-tasks:

1. **Understanding the data:** Start with plotting the underlying pairwise-correlation between the variables.
2. **Effect of batch size:** Implement vanilla (without regularization) least-squares based regression using *a*) Stochastic Gradient Descent (SGD) and *b*) Batch Gradient Descent[1]. Report the weights in each strategy. Plot the loss curve with respect to different batch sizes. Report the difference in convergence of gradient descent (if any) by varying the batch size.
3. **Effect of Regularization:** In this sub-task, perform minimization of the cost function using *a*) *lasso regression* and *b*) *ridge regression* and learn the weights using SGD. Determine the value of the regularization parameter ($\lambda$) in both the cases. Plot/Report the weights in both the cases. Is there any intuitive difference?
4. **Feature Engineering** Keeping subtask 1 in consideration, can you improvise the regularized version of the cost function? You are allowed to experiment with your own cost function (other than sum-of-squares error). If yes, report the weights in this improved version.
5. **Effect on sample distribution:** Train the regression model (without any regularization) using SGD taking 10%, 20%, 30%, 40% and 50% samples (randomly drawn out of the dataset) for training. Plot/Report the test accuracy in each sampled subsets using the testset given. What do you observe when using such a truncated population distribution?
6. Estimate the variance in the noise. Report the strategy for estimation.
7. **Optimal weights:** Using the empirical approach and the intuition you have developed so far, use a scheme (of your choice) to find the most optimal weights to the regression function for this dataset. This may include adding other penalty terms to some weights, or customizing the cost function, etc. Every solution is acceptable as long as it is within the bounds of the definition of **multivariate linear regression**. Report the weights using the strategy that you have followed. You are encouraged to try cross-validation, hyperparameter tuning and regularization to improve the results.

The dataset for Part 1 is available here

## Part 2: Multivariate Logistic Regression (6 points)

This experiment will lead to a practical exercise of classifying handwritten digits across two classes (2 or 9). The data for this experiment has been taken from the commonly used MNIST dataset. In MNIST, you have 28x28 images for labels 0-9. Hint: Here you will need to modify the decision rule where you would label a digit as “2” if $w \times x + b \geq 0$ and vice versa. (You should be able to see why this threshold holds). Make use of the sigmoid activation function here:

$$f(z) = \frac{1}{1 + e^{-z}}$$

Also, use log-loss for this task defined as follows:

$$\frac{-1}{N} \sum_n (y_n \log \hat{y_n} + (1 - y_n) \log(1 - \hat{y_n}))$$

Firstly, download the entire MNIST dataset from the following link: data.

[1] An algorithmic demonstration of SGD and Batch GD is available here

Note that in the above link, there is a *get_data.py* file which contains a boilerplate code for reading the MNIST data. For this part (part 2), use the X_subset and y_subset numpy arrays. Now perform the following sub-tasks:

1. **Effect of batch size:** Implement vanilla (without regularization) logistic regression using *a*) Stochastic Gradient Descent (SGD) and *b*) Batch Gradient Descent. Report the weights in each strategy. Plot the loss curve with respect to different batch sizes. Report the difference in convergence of gradient descent (if any) on by varying the batch size.
2. **Effect of Regularization:** In this sub-task, perform minimization of the cost function using *a*) *lasso regression* and *b*) *ridge regression* and learn the weights using SGD. Determine the value of the regularization parameter ($\lambda$) in both the cases. Plot/Report the weights in both the cases. Is there any intuitive difference?
3. Report the highest accuracy that you are able to achieve. Try to visualize some samples where the model prediction is going wrong. Do you see any pattern here?
4. **Optimal weights:** Using the empirical approach and the intuition you have developed so far, use a scheme (of your choice) to find the most optimal weights to the regression function for this dataset. This may include adding other penalty terms to some weights, or customizing the cost function, etc. Every solution is acceptable as long as it is within the bounds of the definition of **multivariate logistic regression**. Report the weights using the strategy that you have followed.

## Part 3: BONUS (4 points)

For the bonus task, you won't be given the specifics other than extending the work done in Part 2 to work with the whole MNIST dataset (to classify across all digits 0-9). Write about your approach in the report along with the experimentation and the final accuracy that you are able to get on the test set. You can also plot a confusion matrix to showcase the performance of your model.

Do note that you are only allowed to use Linear models for this task. Don't get into complex models like SVM and Neural Networks yet :)

## Code submission format

Apart from the viva, there will be an autograded component to the evaluation of this assignment. Hence it is necessary to follow the exact guidelines for this part.

- Your submission should be a .zip file which upon decompressing should yield a folder with the name as your entry numbers (separated with an underscore character). Inside this folder there should be your codes/scripts for all the parts, named as `p1s1.py, p1s2.py, p1s3.py, p1s4.py, p1s5.py, p1s6.py, p1s7.py, p2s1.py, p2s2.py, p2s3.py, p2s4.py, and p3.py`.
- Specifically, there must be a code script for Part1 (subpart 7) named as `p1s7.py`; code script for Part 2 (subpart 4) named as `p2s4.py`; code script for Part 3 named `p3.py`. These three scripts will be autograded.
- Along with the scripts, there should exist a report file named `report.pdf`. Feel free to add a `requirements.txt` file.
- For an illustration of how your submission should look like, a basic composure will be available here (when your entry numbers in the group are e.g. 2020CSZ1234, 2020EE14567, and 2020CS15432).

- **Libraries allowed:** Standard Python libraries for I/O (argparse, csv) data-visulization (matplotlib, seaborn), data-handling (pandas, numpy). External libraries are not allowed. Needless to say, all the code needs to be written from scratch. If any additional libraries are allowed, it will be announced in due time. For consistency, use Python 3.7. There is no need to add the datasets to the submission. We train your code on the original train set and will test it on held-out hidden dataset.
- Also, note that we will be setting a hard limit of maximum time your code is allowed to execute which will be announced soon. Try to optimize your code as much as possible by using vector operations, wisely choosing the learning rate of gradient descent and limiting the number of epochs (also, read about early stopping).

## For part 1

We will run the following command for evaluating this part:

```
# python p1s7.py --train train.csv --test test.csv --out preds.txt
```

Here, we have the following:

- –train : path to the train.csv file
- –test : path to the test.csv file
- –out : path to the file in which predictions will be dumped

In the above example terminal command, the expectation is that your code should train on train.csv file and should use the test.csv file for making the predictions. These predictions need to be dumped in the preds.txt file such that each line contains one prediction.

## For part 2

We will run the following command for evaluating this part:

```
# python p2s4.py --trainX trainX.npy --trainY trainY.npy --testX testX.npy --out preds.txt
```

Here, we have the following:

- –trainX : path to the training image files
- –trainY : path to the training image labels
- –testX : path to the testing image files
- –out : path to the file in which predictions will be dumped

In the above example terminal command, the expectation is that your code should train on trainX.npy and trainY.npy files and should use the testX.npy file for making the predictions. These predictions need to be dumped in the preds.txt file such that each line contains one prediction.

## For part 3

We will run the following command for evaluating this part:

```
# python p3.py --trainX trainX.npy --trainY trainY.npy --testX testX.npy --out preds.txt
```

Here, we have the following:

- –trainX : path to the training image files
- –trainY : path to the training image labels

- –testX : path to the testing image files
- –out : path to the file in which predictions will be dumped

In the above example terminal command, the expectation is that your code should train on trainX.npy and trainY.npy files and should use the testX.npy file for making the predictions. These predictions need to be dumped in the preds.txt file such that each line contains one prediction.