ML Assignment 2 — Logistic Regression

Anirudh Agrawal	Aviral Agarwal	Vikramjeet Das
2018A7PS0099H	2018A7PS0192H	2018A7PS0280H

1 Introduction

In this assignment, we implemented Logistic Regression for binary classification. When a new testing example comes we calculate its probability and predict its class.
In this report, we discuss the methods that we implemented - the pre-processing, the two different gradient descent, their implementations and results.

2 Pre-processing

2.1 Train-test split

As part of training we made 10 different 70% train and 30% test splits of our data and trained or model on them.

2.2 Normalization

We pre-process our training data by scaling each feature to an interval of [0, 1]. This is essentially Min-Max Scaling. We transform each feature in the following way:

$$x_j^{(i)} := \frac{x_j^{(i)} - min(x_j)}{max(x_j) - min(x_j)}$$

where $i = 1...m$ represents each training example, and $j = 1...n$ represents each feature. At a later stage, this helps us get insights into the importance of each feature for our final prediction. It enables us to directly compare each coefficient since all the features are now scaled to the same range.

Next, we apply the same transformation to the test and validation dataset, using the minimum and maximum of each feature as calculated from the train dataset. The transformation for the test dataset is given by:

$$x_{j_{test}}^{(i)} := \frac{x_{j_{test}}^{(i)} - min(x_{j_{train}})}{max(x_{j_{train}}) - min(x_{j_{train}})}$$

A similar transformation is performed for the validation dataset. Using the minimum and maximum from the train dataset is essential here, so we do not end up using data from the test set in any way to evaluate on the test dataset.

3 Models

In logistic regression, the probability for an example belonging to ground truth class is given by:

$$P(Y = y) = (\hat{y})^y (1 - \hat{y})^{1-y}$$

$$\text{If } P(Y = y) = \begin{cases} \geqslant 0.5 & \hat{y} = 1 \ (positive\ example) \\ < 0.5 & \hat{y} = 0 \ (negative\ example) \end{cases}$$

where,

$$\hat{y} = \frac{1}{1 + e^{-(\theta_1 * z_1 + \theta_2 * z_2 + ... + \theta_n * z_n)}}$$

Here, $(y \in 0, 1)$, z_i represents the i^{th} feature vector components for the example and $\theta_0, \theta_1, \theta_2 \ ... \ \theta_n$, or alternatively, for θ, represents parameters. We need to find those parameters that would maximise this probability from maximum likelihood estimation, find θ that minimises this expression:

$$\min \ -\sum_{i=1}^{N} (y \ log(\hat{y}) + (1 - y) log(1 - \hat{y}))$$

4 Methods

We can find a solution for θ in above using two different methods Gradient Descent ans Stochastic Gradient Descent. We provide a short recap of these here.

4.1 Gradient Descent

Gradient descent involves the following algorithm

Algorithm 1: Gradient Descent

Result: Parameter θ minimizing MSE Loss

$\theta = \overrightarrow{0}$;

while *epoch* $<$ *max_epochs* **do**

/* Calculate the loss w.r.t. the whole training set */

$$J(\theta) = -\sum_{i=1}^{N}(y\ log(\hat{y}) + (1-y)log(1-\hat{y}))$$

/* Calculate the gradient of the loss function w.r.t each parameter */

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_{i=1}^{N}(\hat{y} - y) * z$$

/* Update value of each parameter */

$$\theta_j := \theta_j - \alpha \frac{1}{m}\left(\frac{\partial J(\theta)}{\partial \theta_j}\right)$$

/* Break if change in loss from previous epoch is less than a threshold */

if $\Delta loss < \epsilon$ **then**

break;

4.2 Stochastic Gradient Descent

Stochastic gradient descent involves the following algorithm

Algorithm 2: Stochastic Gradient Descent

Result: Parameter θ minimizing MSE Loss

$\theta = \overrightarrow{0}$;

while *epoch* $<$ *max_epochs* **do**

foreach *training instance* $x^{(i)}$ **do**

/* Calculate the loss w.r.t. each training example individually */

$$J(\theta) = -(y\ log(\hat{y}) + (1-y)log(1-\hat{y}))$$

/* Calculate the gradient of the loss function w.r.t each parameter */

$$\frac{\partial J(\theta)}{\partial \theta_j} = (\hat{y} - y) * z$$

/* Update value of each parameter */

$$\theta_j := \theta_j - \alpha\left(\frac{\partial J(\theta)}{\partial \theta_j}\right)$$

/* Break if change in loss from previous epoch is less than a threshold */

if $\Delta loss < \epsilon$ **then**

break;

5 Performance Metrics

- **Loss**: It is the amount of uncertainty in our prediction based on how much it varies from the actual label.
- **Accuracy**: It is the ratio between number of correct prediction made and total predictions made.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- **Recall**: It tells us about the proportions of actual positive predicted by our model.

$$Recall = \frac{TP}{TP + FN}$$

- **Precision**: It tells us about the proportion of correct predictions returned by our model.

$$Precision = \frac{TP}{TP + FP}$$

- **F1 score**: It is harmonic mean of precision and recall, it helps us to adjust importance of one over the other.

6 Results

We provide the plots for the accuracy of GD and SGD as training progresses for three different values of learning rate, 0.1, 0.01 and 0.001. We also include the train and test performance metrics, the loss, accuracy, precision, recall and F1 score.

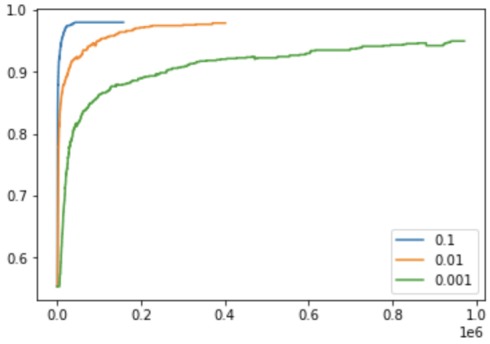

Figure 1: Plots of accuracy for Gradient Descent

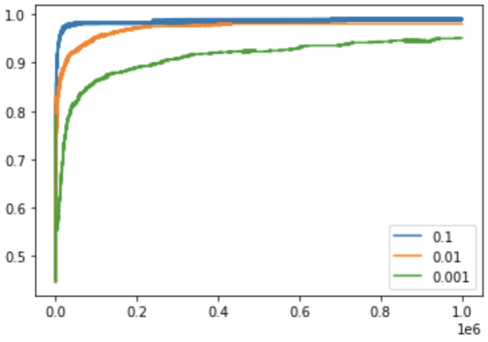

Figure 2: Plots of accuracy for Stochastic Gradient Descent

Metric	GD		SGD	
	Train	Test	Train	Test
Loss	0.0802	0.0800	0.0546	0.0542
Accuracy(%)	99.63	99.62	99.78	99.75
Precision	0.9963	0.9962	0.9978	0.9975
Recall	0.9812	0.9829	0.9826	0.9838
F1 Score	0.9887	0.9895	0.9901	0.9906