[CMPUT 466/566] Machine learning

Coding Assignment 2

Submission Information
The student should submit a zip file containing a pdf report and all the code which should replicate the results.

Download: Codebase

Problem 1 [50%]

In this problem, you are asked to apply linear regression and logistics regression for binary classification. In particular, this problem shows that linear regression is a bad model for classification problems.

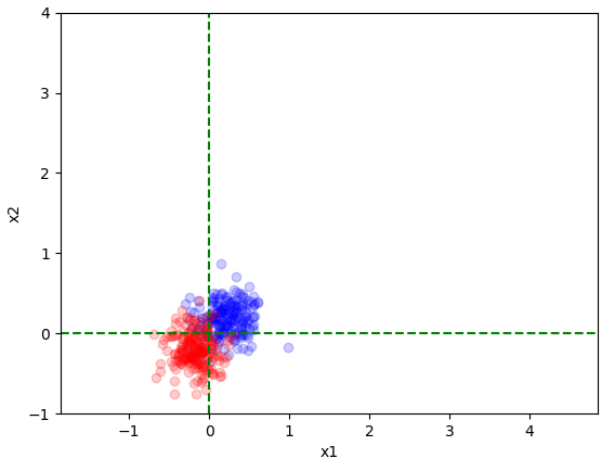
We consider a binary classification problem, where the input is a two-dimensional vector and the output is {0,1}. In other words, $\boldsymbol{x} \in \mathbb{R}^2$ and $t \in \{0, 1\}$. Specifically, we would train our classifiers on the following two datasets:

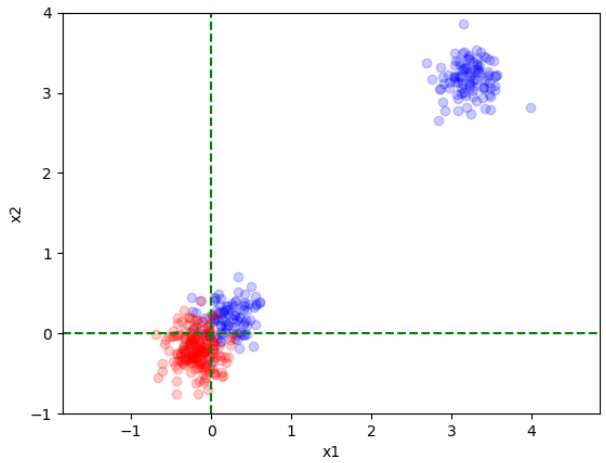
Dataset A:

In this dataset, positive samples are generated by a bivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}_1 = (0.2, 0.2)$ and $\boldsymbol{\Sigma} = \mathrm{diag}(0.2^2, 0.2^2)$. Negative samples are generated by another $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}_2 = (-0.2, -0.2)$. The covariance is the same as the positive samples.

We have 400 samples in total, where 200 are positive and 200 are negative. The dataset is plotted in the left panel below.

Dataset B: We now construct a new dataset by shifting **half** of the positive (blue) samples to the **upper right** as shown in the right panel. In other words, the positive samples are generated by $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$ and $\mathcal{N}(\boldsymbol{\mu}_1', \boldsymbol{\Sigma})$ with equal probability, where $\boldsymbol{\mu}_1' = (3.2, 3.2)$.





Dataset A Dataset B

Questions:

For both Dataset A and Dataset B:

(1) Train a classifier by thresholding a linear regression model. In other words, treat the target 0/1 labels as real numbers, and classify a sample as positive if the predicted value is greater than or equal to 0.5.

(2) Train a logistic regression classifier on the same data.

Note: you can use either the closed-form solution or the (stochastic) gradient descent method. If you use gradient descent, let the learning rate be 0.1 and the number of epochs be at least 1000. For the stochastic version, batch size can be any number that works properly.

Submission (four numbers and four plots)**:**

Report the following four numbers

1. Training accuracy of **linear** regression on **Dataset A**
2. Training accuracy of **logistic** regression on **Dataset A**
3. Training accuracy of **linear** regression on **Dataset B**
4. Training accuracy of **logistic** regression on **Dataset B**

and plot four decision boundaries. Please make clear which classifier is applied in the plot.

Problem 2 [50%]

In this coding problem, we will implement the softmax regression for multi-class classification using the MNIST dataset. (If you have issues downloading the dataset, please consider trying again with other browsers.)

Dataset

First, download the datasets from the link above. You need to unzip the .gz file to the code folder by either double clicking or some command like `gunzip -k file.gz`

The dataset contains 60K training samples, and 10K test samples. Again, we split 10K from the training samples for validation. In other words, we have 50K training samples, 10K validation samples, and 10K test samples. The target label is among {0, 1, …, 9}.

Algorithm

We will implement stochastic gradient descent (SGD) for cross-entroy loss of softmax as the learning algorithm. The measure of success will be the accuracy (i.e., the fraction of correct predictions).

The general framework for this coding assignment is the same as SGD for linear regression, so you may re-use most of the code. However, you shall change the computation of output, the loss function, the measure of success, and the gradient whenever needed.

Implementation trick

For softmax classification, you may encounter numerical overflow if you just follow the equation mentioned in the lecture.

$$z_i = \boldsymbol{w}_i^\top \boldsymbol{x} + b_i$$
$$y_i = \frac{\exp\{z_i\}}{\sum_j \exp\{z_j\}} = \frac{\exp\{\boldsymbol{w}_i^\top \boldsymbol{x} + b_i\}}{\sum_j \exp\{\boldsymbol{w}_j^\top \boldsymbol{x} + b_j\}}$$

The observation is that the exp function increases very fast with its input, and very soon exp(z) will give NAN (not a number).

The trick is to subtract every z_i by the maximum value $z_1, \cdots, z_K$.

In other words, we compute

$\tilde{z}_i = z_i - z_{\max}$, where $z_{\max} = \max_{k=1}^{K} z_k$, and then we have

$$y_i = \frac{\exp\{\tilde{z}_i\}}{\sum_j \exp\{\tilde{z}_j\}}$$

Note that the gradient is computed with y, and since subtracting a constant before softmax doesn't affect y, it doesn't affect the gradient either.

Without changing the the default hyperparameters, we report three numbers:

1. The number of epoch that yields the best validation performance,
2. The validation performance (accuracy) in that epoch, and
3. The test performance (accuracy) in that epoch.

and two plots:

1. The learning curve of the training cross-entropy loss, and
2. The learning curve of the validation accuracy.

Note: a numerical error (i.e., -log(0)) may occur when plotting the loss. In that case you can truncate the loss to -log(1e-16).