CMPE462 - Project I
Logistic Regression

Sadullah Gültekin - 2014400066
Bahadır Hocamoğlu - 2014400117
Yusuf Başpınar - 2014400042

Spring Term, 2020

1 Introduction

In this project we implemented Logistic Regression from scratch. We used our Logistic Regression implementation to differentiate images of handwritten digit 1 and 5 which are provided from handwritten digit data set called MNIST. In this file we will first show you how we implemented the gradient of the logistic loss with respect to w. After that derivation of gradient function, training results are shown. In the end comments on each tasks (feature extraction, model training, and evaluation) and plots of results can be seen.

In the whole code we used 3 different data representations, first two (representation 1, representation 2) consists of 2 features in order to be able to plot data set. Third one contains 7 features which makes it difficult to plot. That is why, we gave only numerical result regarding representation 3.

2 Comments

2.1 Comments on Task 1 (Feature Extraction)

In this task we first extracted features seven features from the data, and used different sets of them to create three different representations. Below we will explain how did we extract these seven features:

- ***Symmetry:*** We computed the symmetry feature as the negative of the norm of the difference between the image and its y-axis symmetry.
- ***Average intensity:*** We computed the intensity as the average pixel value of the image.
- ***Cropped intensity:*** Same as average intensity, however considers only the middle part of the image
- ***Average row length:*** We computed the average lenght of non black lines in each row. In each row, there are white and black pixels. For each row, we got the first and last non-black pixels, and subtracted their index number.
- ***Centrality:*** For each row, we computed how much the pixels are gathered toward center. For each row, we computed a numerical value which increases if the white pixels get closer to the edges.
- ***Changes in columns:*** For each column we calculated the number of peak changes. In other words, we found color changes for each column. What we expect is, for five images, this number should be higher.
- ***Number of white pixels:*** This feature is similar to average intensity feature, but this is only the number of white pixels.

- ***Representation 1:***
  - ***Symmetry***
  - ***Average intensity***
- ***Representation 2:***
  - ***Centrality***
  - ***Changes in columns***
- ***Representation 3:***
  - ***Symmetry***
  - ***Average intensity***
  - ***Centrality***
  - ***Changes in columns***
  - ***Cropped intensity***
  - ***Average row length***
  - ***Number of white pixels***

2.2 Comments on Task 2 (Logistic Regression)

We implemented logistic regression as a single class which has the fit and predict functions that can be used.

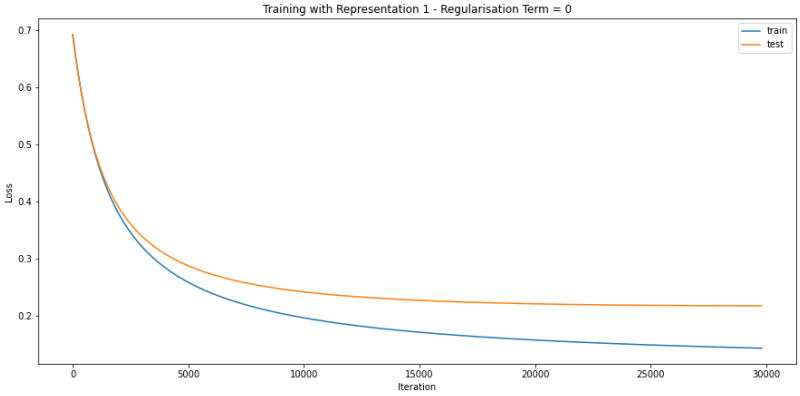
There are two similar logistic regression implementations. The difference is their loss functions. One of them (HardToMinimizeLogisticRegression) uses a loss function that is comparatively difficult to minimize. That is why for this code, we used the other implementation.

In the creation of logistic regression object, one needs to give number of features to use as a parameter in order to create model that has a compatible shape with the data. In the initialization code, we initialized the weights of the model using gaussian normal distribution with 0 mean and 0.1 variation, and multiplied with 0.001 to have small weights before the training.

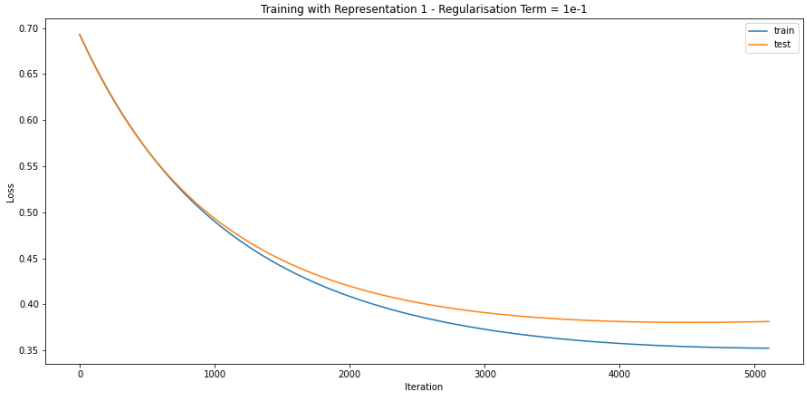
Our fit function takes test data optionally. This may be misunderstood like the test data is being used in the training. However it is only given to calculate test loss in each iteration. At the end, when we plot out loss values, we can also use test data loss to see whether the model is over fitting or not.

2.3 Comments on Task 3 (Evaluation)

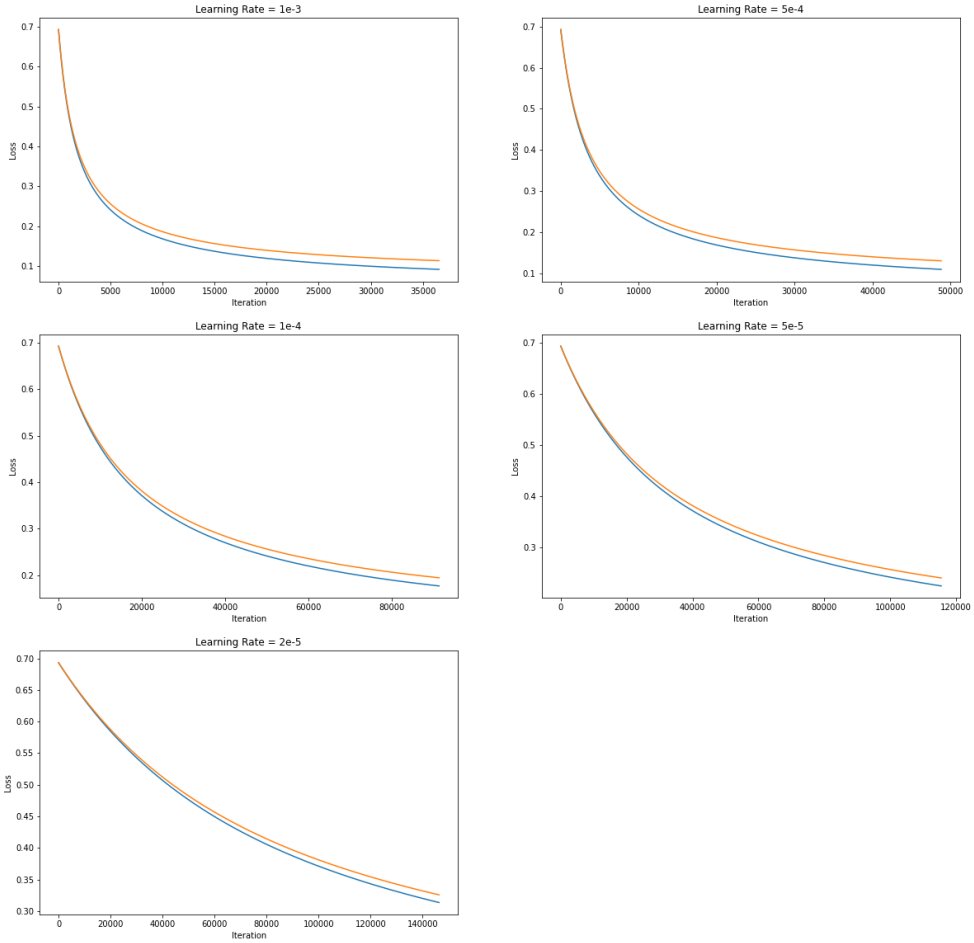
To evaluate our implementation, we performed various experiments. At first, after the implementing the logistic regression, we tried to perform a basic training to see whether the model is able to learn or not. Below is the learning curve of the model:



Then, to test the regularization we add 0.1 regularization. What we expected is a decrease in the difference between the train and test losses. As we can see from the curve above, regularization also works fine.



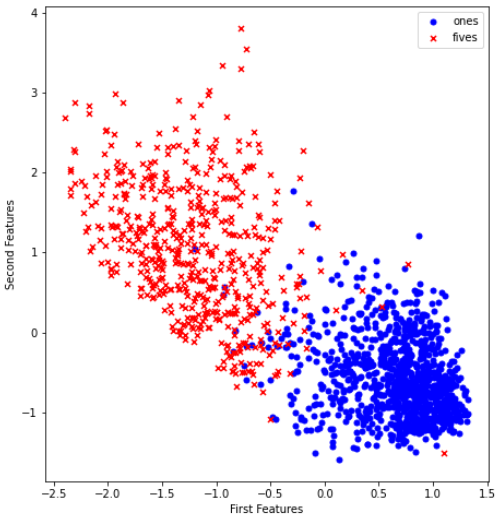
Another aspect we need to check is the learning rate aspect. In a training, we expect to see faster convergence with high learning rates. However, there is a risk with high learning rate, which may not converge to the best point. To see the learning rate effect we run logistic regression with learning rates 1e-3, 5e-4, 1e-4, 5e-5, 2e-5. As we expected lower learning rates converges slower. Learning curves are shown below:

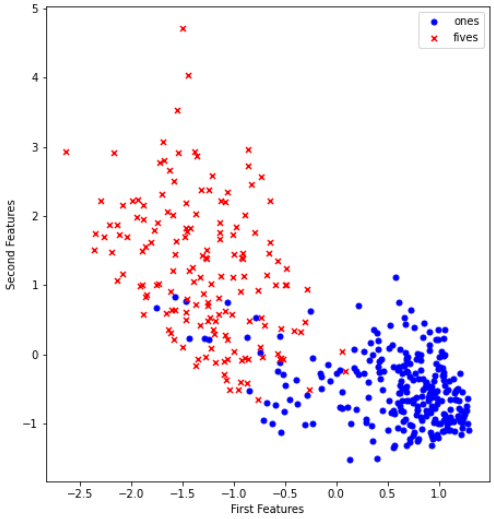


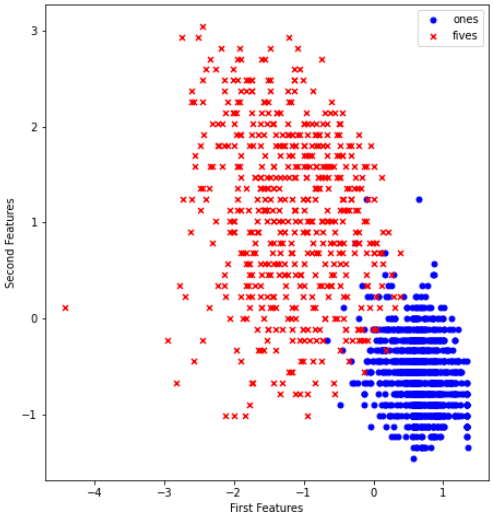
Other test results can be found at the sections 4 (Training results for different representations) and 5 (Cross validation results for different representations)

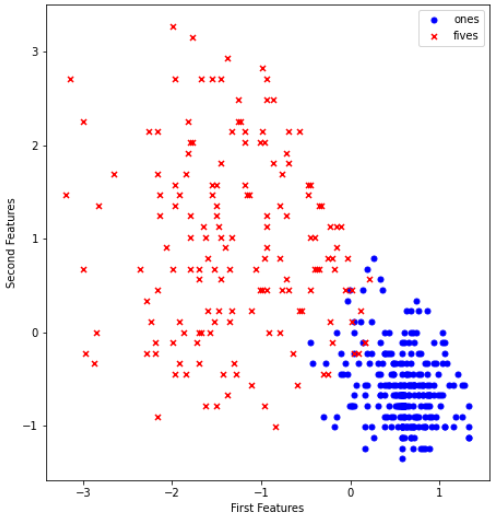
For now, we stop when our loss changes with an amount less then a threshold. To increase the test accuracy, we can decrease this threshold which will increase the accuracy probably. However training time will also increase. Another approach may be using learning rate which decreases by time. If we run our algorithm long enough, our loss will converge and fluctuate around the same loss value. When we observe a loss trend like this, we can decrease the learning rate which may lead to lower loss values and higher test accuracy.

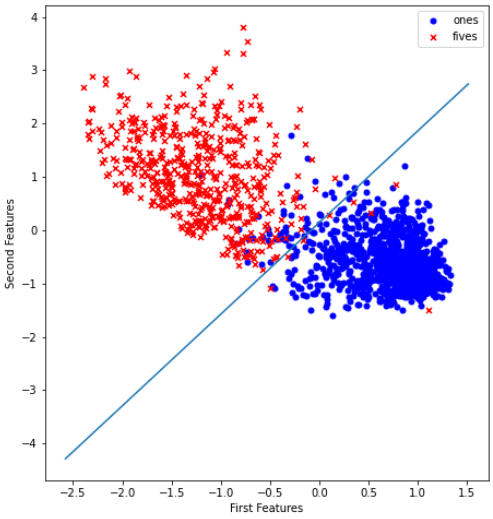
We have 7 features and 3 different representations that contains different features in it. These features are explained in the previous section. Here we will give scatter plots, that discriminates the classes. First one is the representation 1 and second one is the representation 2. We can observe that second one is more discriminative and gives higher accuracy. Finally we gave lines that our trained model draws to discriminates between two classes.

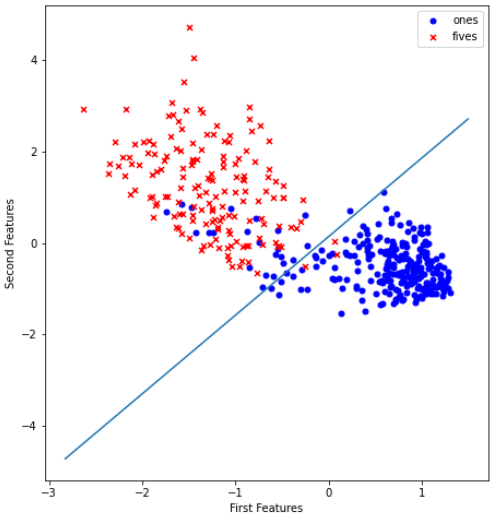


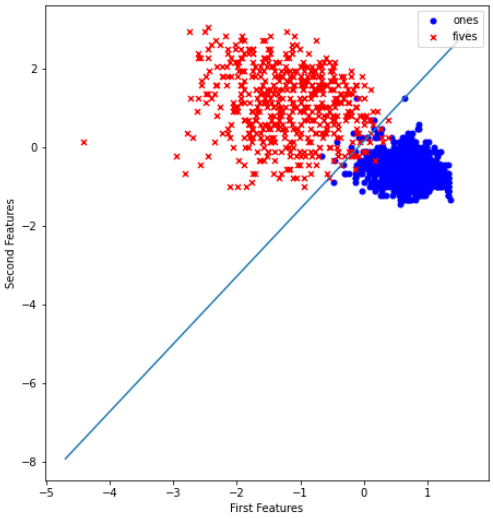


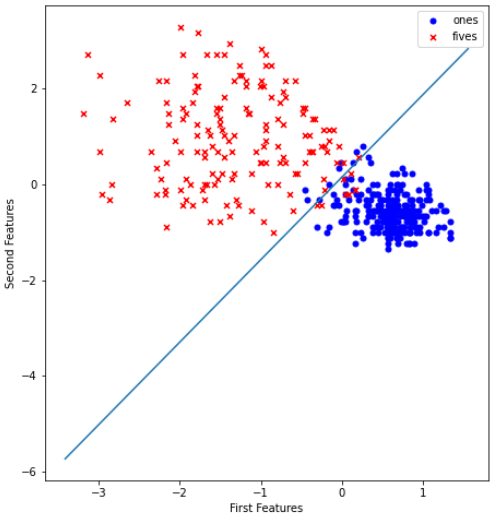












3 Derivation of Gradient

$L = \frac{1}{N}\sum_{n=1}^{N}(1 + exp(-y_n W^t X))$

$\frac{\partial L}{\partial W} = \frac{\partial}{\partial W}(\frac{1}{N}\sum_{n=1}^{N}(1 + exp(-y_n W^t X)))$

$\frac{\partial L}{\partial W} = \frac{1}{N}\sum_{n=1}^{N}\frac{\partial}{\partial W}(1 + exp(-y_n W^t X))$

$\frac{\partial L}{\partial W} = \frac{1}{N}\sum_{n=1}^{N}\frac{-y_n X exp(-y_n W^T X)}{1+exp(-y_n w^T X)}$

4 Results of Trainings

Representation	Regularization Term	Training Accuracy	Test Accuracy
Representation 1	0.0	0.957078795643818	0.9363207547169812
Representation 1	0.1	0.9551569506726457	0.9386792452830188
Representation 2	0.0	0.9807815502882767	0.9575471698113207
Representation 2	0.1	0.9788597053171044	0.9575471698113207
Representation 3	0.0	0.9903907751441384	0.9669811320754716
Representation 3	0.1	0.9891095451633568	0.9622641509433962

5 Results of Cross Validation

Representation 1		
Regularization Term	Accuracy Mean	Accuracy Std
0.4	0.9532419922995002	0.011177729313485641
0.3	0.9526030146637176	0.011845370204363243
0.2	0.9538850659457688	0.012548761026771412
0.1	0.9545260915867944	0.012348706390970285
0.0	0.9570901941508971	0.014371208420494896

Representation 2		
Regularization Term	Accuracy Mean	Accuracy Std
0.4	0.9769435569755058	0.008907939060260842
0.3	0.9769435569755058	0.008907939060260842
0.2	0.9782256082575571	0.008183608566690174
0.1	0.9782256082575571	0.008183608566690174
0.0	0.9807876628164168	0.006382654914751551

Representation 3		
Regularization Term	Accuracy Mean	Accuracy Std
0.4	0.9891128041287786	0.005935822314857168
0.3	0.9891128041287786	0.005935822314857168
0.2	0.9891128041287786	0.005935822314857168
0.1	0.9891128041287786	0.005935822314857168
0.0	0.9884738264929958	0.006585470964556023

Best configuration is training with 0 regularization for representation 1 and 2. For representation 3, best configuration is 0.1 regularization. However, we used 0.1 regularization for all representations to show regularization effects.

Representation	Regularization Term	Accuracy Mean	Accuracy Std
1	0.0	0.9570901941508971	0.014371208420494896
2	0.0	0.9807876628164168	0.006382654914751551
3	0.1	0.9891128041287786	0.005935822314857168