CMPE 462 Machine Learning
Department of Computer Engineering
Bogazici University

PROJECT 1
LOGISTIC REGRESSION

Group Name: Hacı Kolonyası
Student ID1: 2015401183
Student ID2: 2015300084

16.04.2020

Contents

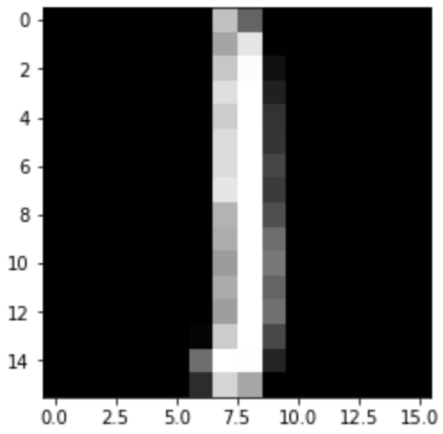
1. Feature Extraction

1.1 Introduction

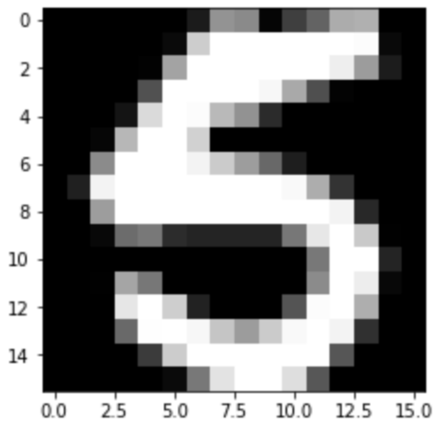
In this section we extracted the necessary features for our models and visualised them. In order to run the project, directory should be changed. Then all cells can be used.

1.2 Display

We displayed one example from each classes resp. hand-written digits 1 and 5.



(a) Hand-written image 1



(b) Hand-written image 5

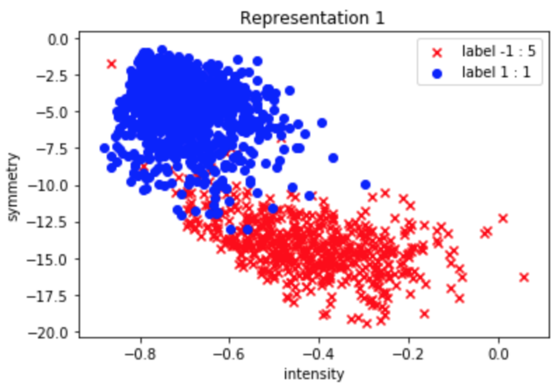
Figure 1.1: Displays

1.3 Implementation of Representation1

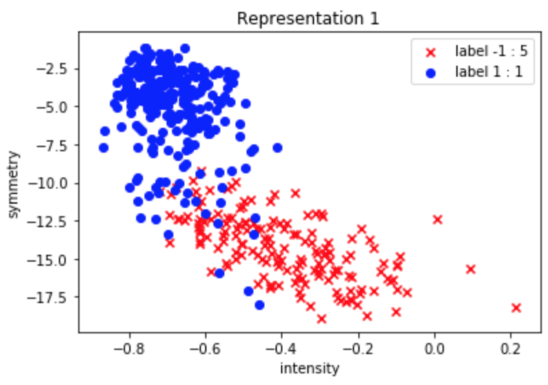
- Here we extracted two features called average intensity and symmetry for both training and test data sets.
- Average intensity is just the mean of the every pixel in one image.
- In order to create symmetry feature, we used numpy flip function to flip our images with respect to y axis.
- Then we computed the negative of the norm of the image and its y axis symmetrical.
- For further detail our jupyter notebook can be checked.

1.3.1 Plots For Representation 1

We displayed our training and test sets with our features. We used intensity features as x axis and symmetry features as y axis



(a) Training data plot



(b) Test data plot

Figure 1.2: Plots for Representation 1

1.4 Implementation of Representation 2

After the first representation, we created another one and called it as Representation 2. It has two features called standard Deviation with respect to y axis for each binary images and blackratios.

1.4.1 Definition of Features for Representation 2

- In order to compute first feature each image is converted to binary image using its pixels values. If a pixel value is positive, it becomes +1, otherwise it becomes 0. These pixels with +1 are indexed according to their columns (The index of the pixels at the first columns is 1, not 0) and the standart deviations are calculated for each image.

$$sample\ image : \begin{bmatrix} -0.3 & 0.8 & -0.7 \\ 0.1 & 0.9 & -0.8 \\ 0 & 0.9 & 0.5 \end{bmatrix} \ binary\ image : \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \ indexed\ image : \begin{bmatrix} 0 & 2 & 0 \\ 1 & 2 & 0 \\ 0 & 2 & 3 \end{bmatrix} \tag{1.1}$$

- so our cluster for this image is 1,2,2,2,3 and their std is 0.7

- On average for digit 1 we expect more white pixels at the center of the image than digit5. So we looked at the two column regions between 2nd and 5th pixel and between 10th and 13th pixel.

- We counted the black pixels in those regions and divided to total number of pixel in those region.

$$\frac{\text{number of black pixels in two regions}}{\text{number of the total pixels in two regions}} \tag{1.2}$$

- For digit 1 we expect that number to be close to 1 and for digit 5 we expect it to be smaller number closer to 0.

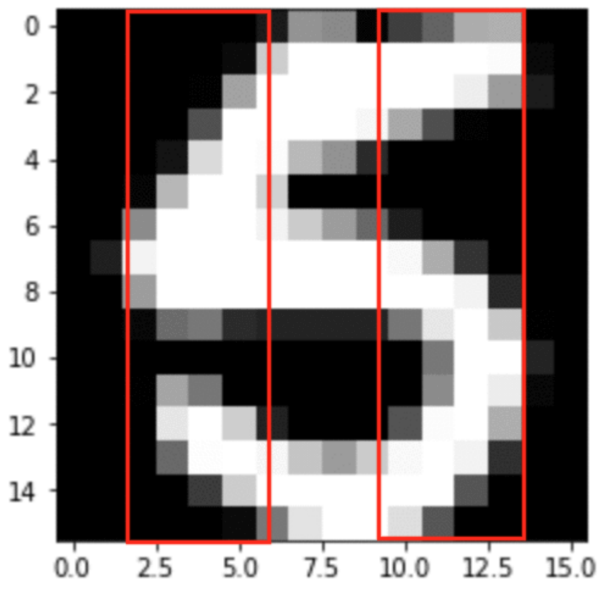
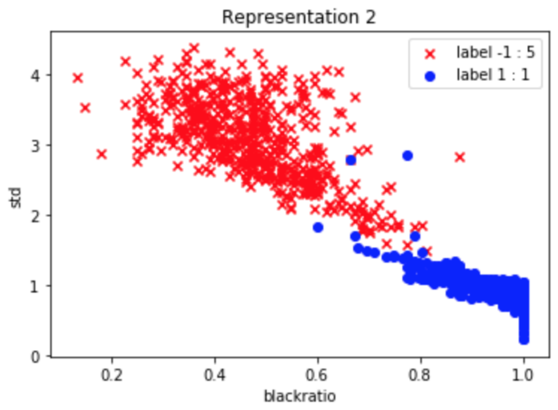


Figure 1.3: An illustration of the regions of interest

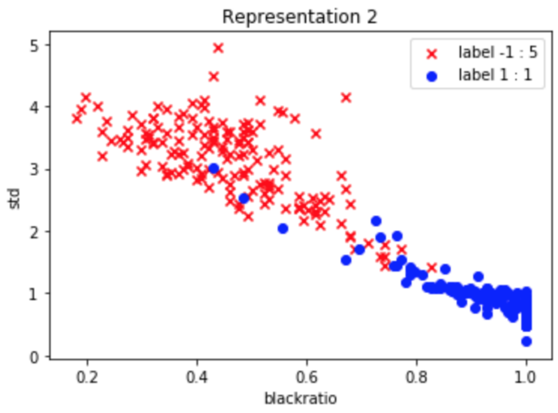
We just counted the black pixels in those regions and divided to all pixels in those regions

1.4.2 Plots For Representation 2

Our plots show that the features which we suggested discriminate data quite well. Black ratios on x axis and std values are on y axis



(a) Training data plot



(b) Test data plot

Figure 1.4: Plots for Representation 2

2. Logistic Regression

2.1 Logistic Regression without Regularization

We implemented the logistic regression classifier from scratch with gradient descent using necessary functions:

- Logistic Loss Function
- Gradient of Logistic Loss Function
- Sigmoid Function
- Hypothesis Function
- Estimation Function
- Accuracy Function

2.1.1 Function Definitions

- Logistic Loss Function (Reference [1])

$$E(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y_n w^T x_n}\right) \tag{2.1}$$

- Gradient of Logistic Loss Function

$$\begin{aligned}\frac{\partial E(w)}{\partial w} &= \frac{\partial}{\partial w}\frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y_n w^T x_n}\right)\\ &= \frac{1}{N}\sum_{i=1}^{N} \frac{\partial}{\partial w}\ln\left(1 + e^{-y_n w^T x_n}\right)\\ &= \frac{1}{N}\sum_{i=1}^{N} \frac{\frac{\partial}{\partial w}\left(1 + e^{-y_n w^T x_n}\right)}{1 + e^{-y_n w^T x_n}}\\ &= \frac{1}{N}\sum_{i=1}^{N} \frac{-y_n x_n e^{-y_n w^T x_n}}{1 + e^{-y_n w^T x_n}}\\ &= \frac{1}{N}\sum_{i=1}^{N} \frac{\frac{-y_n x_n}{e^{y_n w^T x_n}}}{\frac{e^{y_n w^T x_n}+1}{e^{y_n w^T x_n}}}\\ &= \frac{1}{N}\sum_{i=1}^{N} \frac{-y_n x_n}{e^{y_n w^T x_n}+1}\end{aligned} \tag{2.2}$$

- Sigmoid Function

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

- Hypothesis Function (Reference [2])

$$H(x,w) = S(x^T w) = \frac{1}{1+e^{-x^T w}} \tag{2.4}$$

- Estimation Function

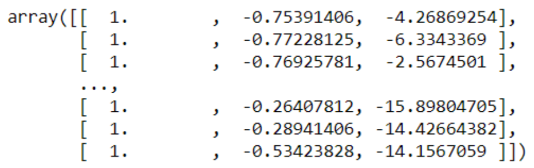
$$Estimation(x,w) = \begin{cases} Label1 & \text{if } H(x,w) > 0.5 \\ Label-1 & \text{if } H(x,w) <= 0.5 \end{cases} \tag{2.5}$$

- Accuracy Function

$$accuracy = \frac{\#correct\ estimation}{\#total\ estimation} \tag{2.6}$$

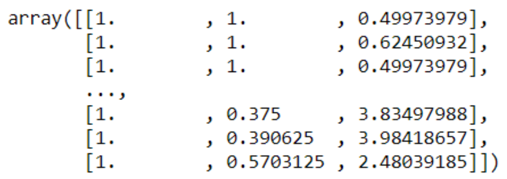
2.1.2 Implementation for Representation 1 & 2

After feature extraction, we concatenated 1 to our features(x1 and x2) for the intercept term (Fig. 2.1a and Fig. 2.1b).



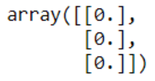
```
array([[  1.        , -0.75391406,  -4.26869254],
       [  1.        , -0.77228125,  -6.3343369 ],
       [  1.        , -0.76925781,  -2.5674501 ],
       ...,
       [  1.        , -0.26407812, -15.89804705],
       [  1.        , -0.28941406, -14.42664382],
       [  1.        , -0.53423828, -14.1567059 ]])
```

(a) x1 (1561x3)



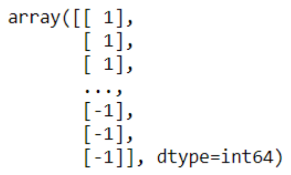
```
array([[1.        , 1.        , 0.49973979],
       [1.        , 1.        , 0.62450932],
       [1.        , 1.        , 0.49973979],
       ...,
       [1.        , 0.375     , 3.83497988],
       [1.        , 0.390625  , 3.98418657],
       [1.        , 0.5703125 , 2.48039185]])
```

(b) x2 (1561x3)



```
array([[0.],
       [0.],
       [0.]])
```

(c) initial weight(3x1)

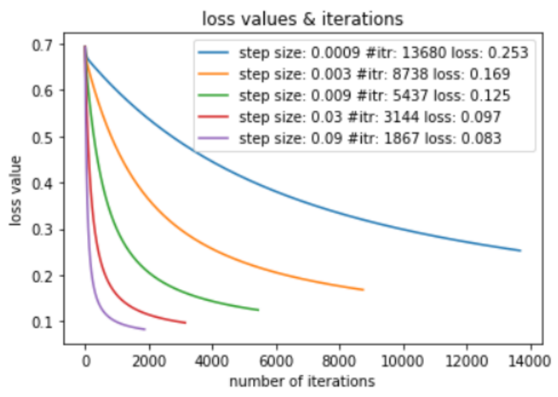


```
array([[ 1],
       [ 1],
       [ 1],
       ...,
       [-1],
       [-1],
       [-1]], dtype=int64)
```

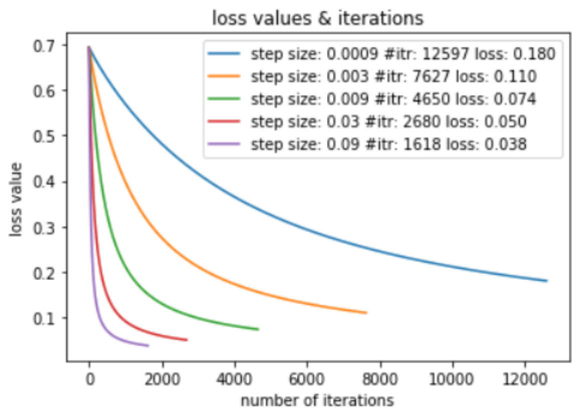
(d) y(1561x1)

Figure 2.1: x, w and y matrices

We implemented the gradient descent algorithm to find the minimum point as Reference [3]. At each iteration, the loss value converged through the minimum loss value. If the difference (we defined as convergence size) previous and current loss value is lower than 10^{-5}, the iteration loop stops. After implementation, we plotted loss functions with respect to iteration count.



(a) Representation 1



(b) Representation 2

Figure 2.2: Loss Values vs Iterations

We observed decreasing loss functions and we figured out that the iterations with lower step size converge to minimum point slowly and their loss values are higher. The best step size is 0.09 for the representation 1 (Fig. 2.2a) and representation 2 (Fig. 2.2b).

When we used much higher step sizes such as 1 to speed up the converge and decrease the loss value, it caused oscillations and it is a problem although it has lower loss values (Fig. 2.3).

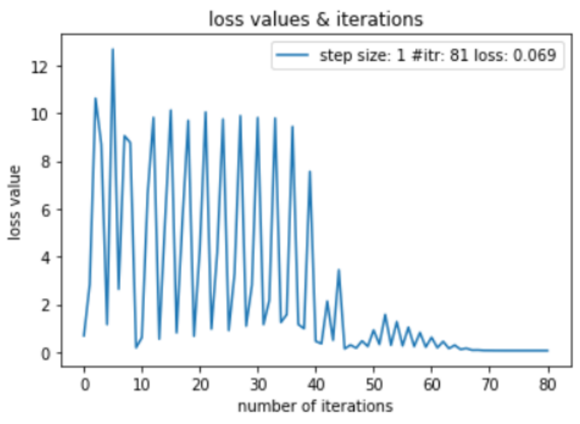


Figure 2.3: Loss Values vs Iterations for Representation 1

Also, we calculated training accuracies to ensure that our implementation works properly. The results can be seen at Table. 2.1 and Tab. 2.2.

Step Size	Accuracy (Training)	Loss Value
0.0009	0.968	0.253
0.003	0.974	0.169
0.009	0.975	0.125
0.03	0.975	0.097
0.09	0.977	0.083

Table 2.1: Representation 1

Step Size	Accuracy (Training)	Loss Value
0.0009	0.994	0.180
0.003	0.993	0.110
0.009	0.994	0.074
0.03	0.993	0.050
0.09	0.995	0.038

Table 2.2: Representation 2

2.2 Logistic Regression with Regularization

We implemented the logistic regression classifier from scratch with gradient descent using necessary functions:

- Logistic Loss Function with Regularization
- Gradient of Logistic Loss Function with Regularization

2.2.1 Function Definitions

- Logistic Loss Function with Regularization We added penalty term to logistic loss function as reference [4] .

$$penalty\ term = \frac{\lambda \|w\|^2}{2} \tag{2.7}$$

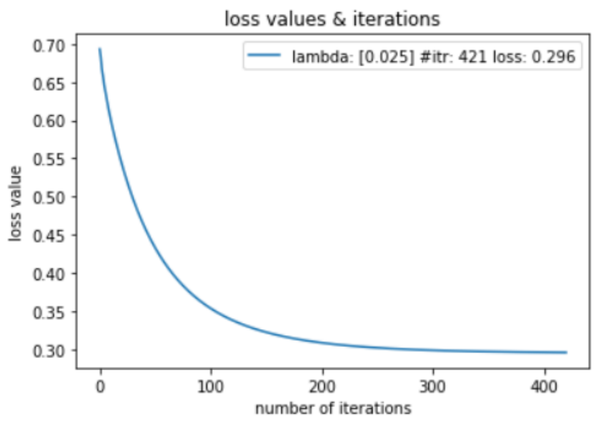
$$E(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y_n w^T x_n}\right) + \frac{\lambda \|w\|^2}{2} \tag{2.8}$$

- Gradient of Logistic Loss Function with Regularization

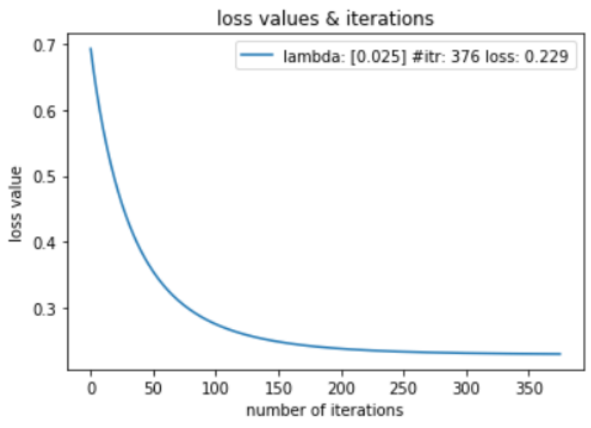
$$derivation\ of\ penalty\ term = \lambda w \tag{2.9}$$

$$\frac{\partial E(w)}{\partial w} = \frac{1}{N}\sum_{i=1}^{N} \frac{-y_n x_n e^{-y_n w^T x_n}}{1 + e^{-y_n w^T x_n}} + \lambda w = \frac{1}{N}\sum_{i=1}^{N} \frac{-y_n x_n}{e^{y_n w^T x_n} + 1} + \lambda w \tag{2.10}$$

2.2.2 Implementation for Representation 1 & 2



(a) Representation 1



(b) Representation 2

Figure 2.4: Loss Values vs Iterations

Similarly previous task, we observed that decreasing loss values for representation 1 and 2 and final loss values are increased due to penalty term.

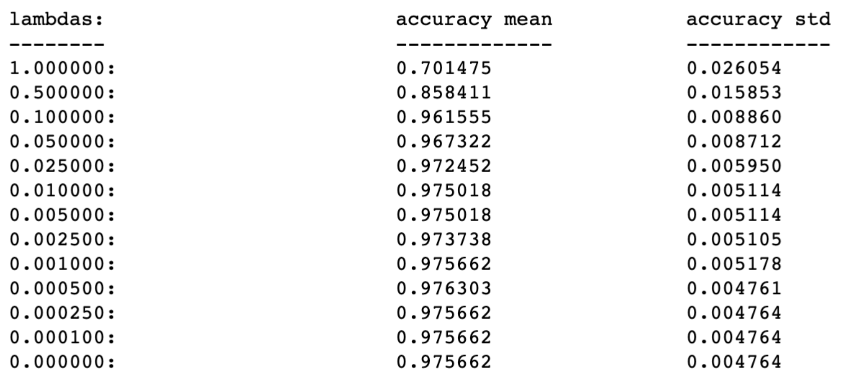
2.3 5-Fold Cross Validation

We implemented a 5-fold cross validation procedure using 13 different lambda values to determine best lambda values for both representations. To check the differences between logistic regression with and without regularization, we used lambda=0 value, too.

2.3.1 Representation 1

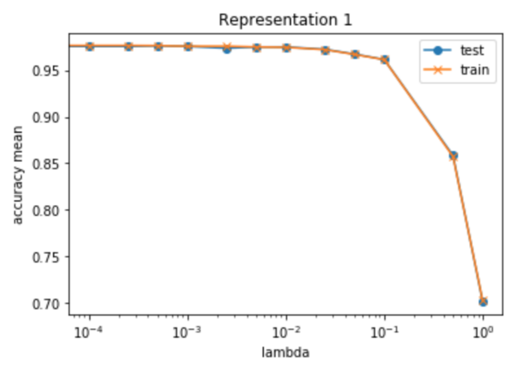
Lambdas	Fold1	Fold2	Fold3	Fold4	Fold5
1	0.69968051	0.68910256	0.73076923	0.66025641	0.7275641
0.5	0.87859425	0.86538462	0.85897436	0.83012821	0.85897436
0.1	0.9744408	0.96794872	0.95833333	0.94871795	0.95833333
0.05	0.97763578	0.97115385	0.95192308	0.96474359	0.97115385
0.025	0.97444089	0.9775641	0.96153846	0.97115385	0.9775641
0.01	0.97124601	0.98076923	0.96794872	0.97435897	0.98076923
0.005	0.97124601	0.98076923	0.96794872	0.97435897	0.98076923
0.0025	0.96805112	0.98076923	0.96794872	0.97435897	0.9775641
0.001	0.96805112	0.98076923	0.97115385	0.9775641	0.98076923
0.0005	0.96805112	0.98076923	0.97435897	0.98076923	0.9775641
0.00025	0.96805112	0.98076923	0.97435897	0.98076923	0.97435897
0.0001	0.96805112	0.98076923	0.97435897	0.98076923	0.97435897
0	0.96805112	0.98076923	0.97435897	0.98076923	0.97435897

Table 2.3: Cross Validation Accuracies vs Lambdas

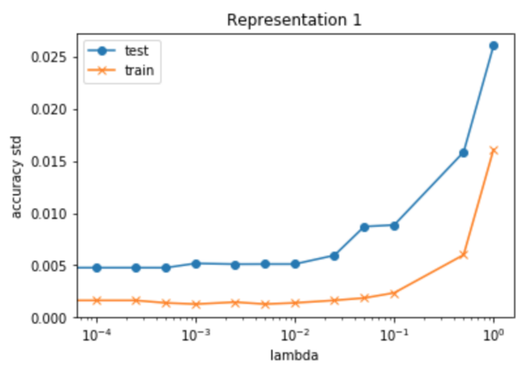


```
lambdas:                        accuracy mean           accuracy std
--------                        -------------           ------------
1.000000:                       0.701475                0.026054
0.500000:                       0.858411                0.015853
0.100000:                       0.961555                0.008860
0.050000:                       0.967322                0.008712
0.025000:                       0.972452                0.005950
0.010000:                       0.975018                0.005114
0.005000:                       0.975018                0.005114
0.002500:                       0.973738                0.005105
0.001000:                       0.975662                0.005178
0.000500:                       0.976303                0.004761
0.000250:                       0.975662                0.004764
0.000100:                       0.975662                0.004764
0.000000:                       0.975662                0.004764
```

Figure 2.5: Cross Validation Accuracies Mean and Std



(a) Accuracy Mean vs Lambdas



(b) Accuracy Std vs Lambdas

Figure 2.6: Representation 1

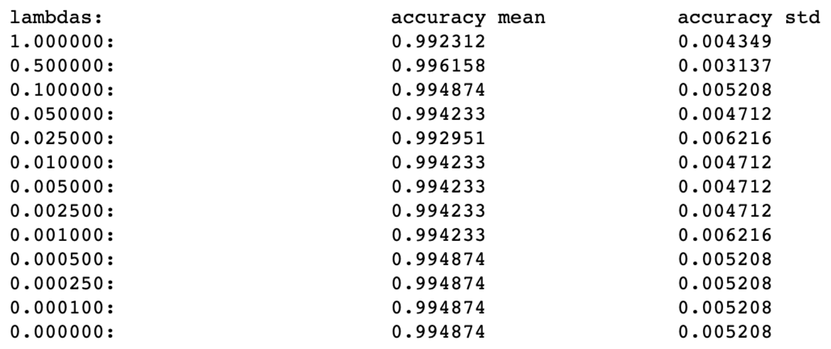
Conclusion:

For the representation 1, Lambda value with 0.0005 is the best lambda with the highest mean and with the lowest std. So we decided to use it instead of others.
As there are not much data points and the data points are shuffled, the results can be changed. If there were much more data points, we could get concrete results.

2.3.2 Representation 2

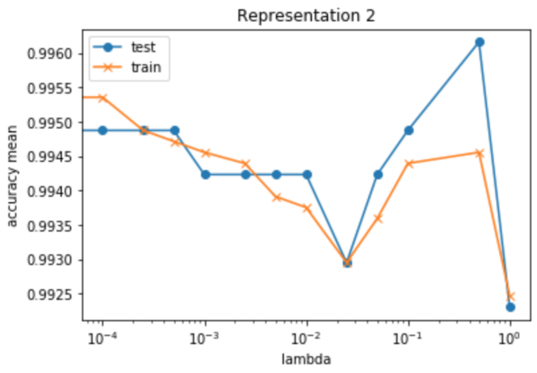
Lambdas	Fold1	Fold2	Fold3	Fold4	Fold5
1	0.99361022	0.99038462	1.00000000	0.99038462	0.98717949
0.5	0.99361022	0.99358974	1.00000000	1.00000000	0.99358974
0.1	0.99680511	0.98717949	1.00000000	1.00000000	0.99038462
0.05	0.99680511	0.98717949	1.00000000	0.99679487	0.99038462
0.025	0.99680511	0.98397436	1.00000000	0.99679487	0.98717949
0.01	0.99680511	0.98717949	1.00000000	0.99679487	0.99038462
0.005	0.99680511	0.98717949	1.00000000	0.99679487	0.99038462
0.0025	0.99680511	0.98717949	1.00000000	0.99679487	0.99038462
0.001	0.99680511	0.98397436	1.00000000	1.00000000	0.99038462
0.0005	0.99680511	0.98717949	1.00000000	1.00000000	0.99038462
0.00025	0.99680511	0.98717949	1.00000000	1.00000000	0.99038462
0.0001	0.99680511	0.98717949	1.00000000	1.00000000	0.99038462
0	0.99680511	0.98717949	1.00000000	1.00000000	0.99038462

Table 2.4: Cross Validation Accuracies vs Lambdas

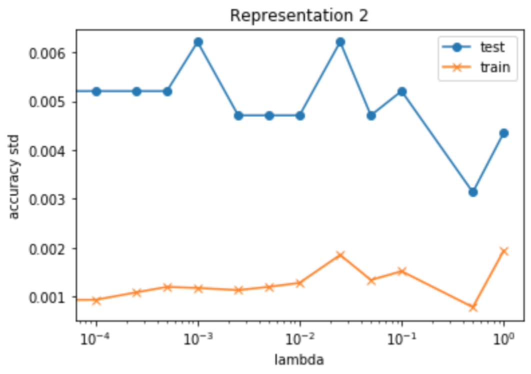


```
lambdas:                        accuracy mean          accuracy std
1.000000:                       0.992312               0.004349
0.500000:                       0.996158               0.003137
0.100000:                       0.994874               0.005208
0.050000:                       0.994233               0.004712
0.025000:                       0.992951               0.006216
0.010000:                       0.994233               0.004712
0.005000:                       0.994233               0.004712
0.002500:                       0.994233               0.004712
0.001000:                       0.994233               0.006216
0.000500:                       0.994874               0.005208
0.000250:                       0.994874               0.005208
0.000100:                       0.994874               0.005208
0.000000:                       0.994874               0.005208
```

Figure 2.7: Cross Validation Accuracies Mean and Std



(a) Accuracy Mean vs Lambdas



(b) Accuracy Std vs Lambdas

Figure 2.8: Representation 2

Conclusion:

For the representation 2, lambda value with 0.5 is the best lambda with highest accuracy and with the least amount of standard deviation. so we will use it instead of any other lambdas. Similarly representation 1, the results can be changed. If there were much more data points, we could get concrete and better results.

3. Evaluation

3.1 Accuracy for Representation 1 & 2

In this part we are going to evaluate the accuracy of our findings from previous sections. We will try to see if we could improve our accuracy of our models with regularization method.

3.1.1 Definition of accuracy

$$\frac{\text{number of correctly classified samples}}{\text{total number of samples}} x100$$

3.1.2 Accuracy for Representation 1 with & without Regularization

- From previous analysis we decided to use learning rate of 0.09. When we plug this this number in our logistic regression function we get a weight vector called w_{best}.

$$w_{best} = [4.7672, -4.1089, 0.7210]$$

- Then we used this weight vector to calculate our train and test accuracy. The corresponding accuracy are:

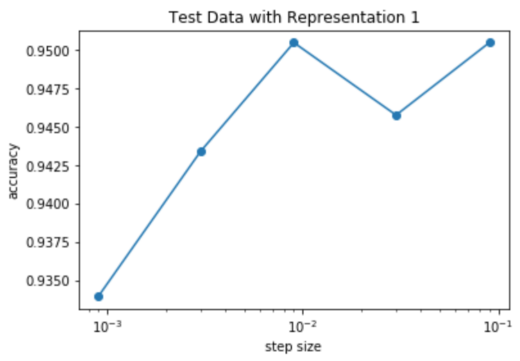
$$\begin{aligned} Training\ \ accuracy &: 97.6937 \\ Test\ \ accuracy &: 95.0471 \end{aligned}$$

- Similarly we calculated w_{lambda} with our lambda choice from 5-Fold CV which is 0.0005 and with best learning rate 0.09.

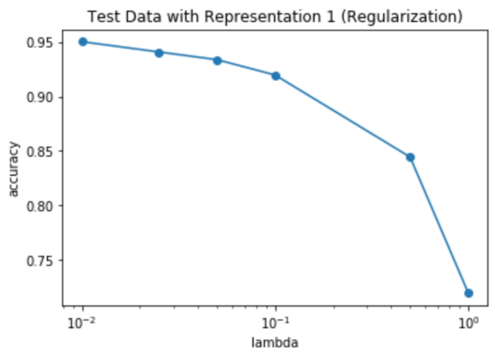
$$w_{lambda} = [4.4274, -3.8171, 0.6724]$$

- Again we used this weight vector to calculate our train and test accuracy. The corresponding accuracy are:

$$\begin{aligned} Training\ \ accuracy &: 97.6297 \\ Test\ \ accuracy &: 94.8113 \end{aligned}$$



(a) Test accuracy for different step sizes



(b) Test data for different lambda with same step size

Figure 3.1: Representation 1

Conclusion:

1. Our results suggest that our best step size outperforms several different step sizes. So this is an evidence that our choice might be the best learning rate among others.

2. At the same time even though we tried to find best lambda from 5-Fold CV to improve our test performance, we could not improve it.

3. Figure 3.1.b suggests that even for other lambda values test accuracy did not improve. So in this model over fitting might not be the problem and regularization does not help to improve accuracy.

3.1.3 Accuracy for Representation 2 with & without Regularization

- Similarly we decided to use learning rate of 0.09 for this implementation as well. When we plug this this number in our logistic regression function we get a weight vector called w_{best2}.

$$w_{best2} = [2.8005, 3.8808, -3.2151]$$

- Then we used this weight vector to calculate our train and test accuracy. The corresponding accuracy are:

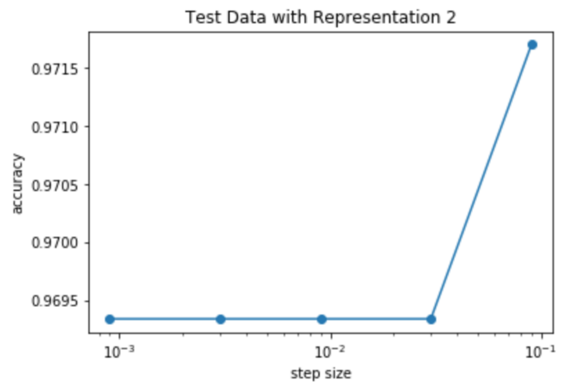
$$Training\ \ accuracy : 99.5515$$
$$Test\ \ accuracy : 97.1698$$

- Similarly we calculated $w_{lambda2}$ with our lambda choice from 5-Fold CV which is 0.5 .

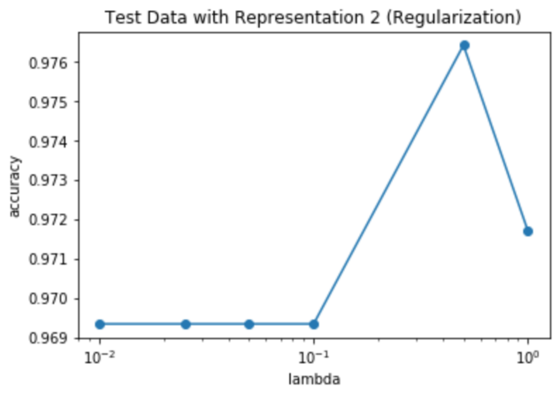
$$w_{lambda2} = [0.2661, 0.3725, -0.3460]$$

- Again we used this weight vector to calculate our train and test accuracy. The corresponding accuracy are:

$$Training\ \ accuracy : 99.487$$
$$Test\ \ accuracy : 97.6415$$



(a) Test accuracy for different step sizes



(b) Test data for different lambda with same step size

Figure 3.2: Representation 2

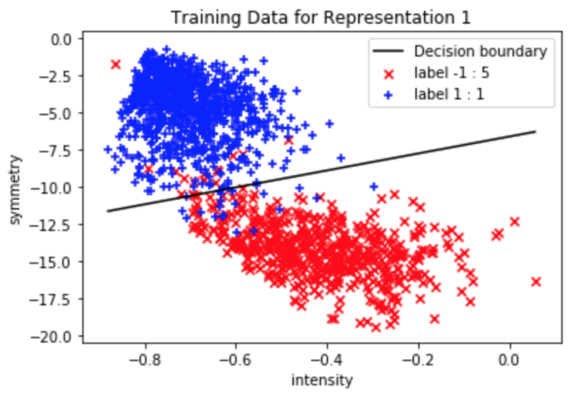
Conclusion:

1. Our step size performs better than the other step sizes which is suggested by Figure 3.2.a.
2. Figure 3.2.b suggests that our choice of lambda (0.5) performs well in comparison to several other lambdas
3. Our training accuracy decreased at the same time our test accuracy increased. This implies that our model for Representation 2 had some over fitting issue but our regularization method helped to improve our models performance.

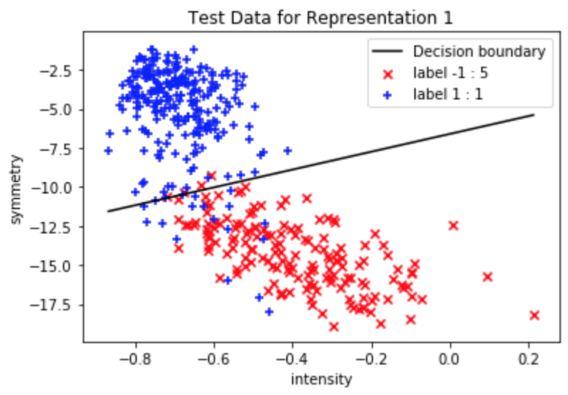
3.2 Decision Boundary

We visualised the decision boundary by the line given by $\mathbf{w}^T x = 0$. We labeled intensity values as $x1$ and symmetry values as $x2$. Since we know w_{best} from previous section we constructed the following equation.

$$x2 = \frac{4.1089}{0.7210}x1 - \frac{4.7672}{0.7210} \tag{3.1}$$



(a) Training data decision boundary



(b) Test data decision boundary

Figure 3.3: Decision boundaries for Representation 1

3.3 Comments on project

1. For Representation 1 it did not improve test accuracy because the features are not really discriminative. Also it did not reduce the gap. Our test accuracy decreased with regularization. This indicates that overfitting might not be the source of problem here. However, for the Representation 2 it increased test accuracy at the same time it decreased the gap between training and test accuracy.

2. Representation 2 gave better results in comparison to Representation 1. We can observe that from test accuracies from the corresponding Represenation plots. Representation 2 is more discriminative.

3. Following might be some ideas to improve our test accuracy

   (a) We can add more features so that our representations will be more discriminative.

   (b) We can add more data points so that our model can see more cases.

   (c) We can conduct error analysis. We can look at the misclassified examples and try to identify the wrongly labeled images and focus on that problem. For example, if one images are more misclassified we can try to add a new feature to solve this issue.

Bibliography

[1] Baytaş, İ., 2020. Logistic Regression Slide.CMPE 462 Machine Learning Bogazici University, p.15.

[2] Baytaş, İ., 2020. Logistic Regression Slide.CMPE 462 Machine Learning Bogazici University, p.17.

[3] Baytaş, İ., 2020. Logistic Regression Slide.CMPE 462 Machine Learning Bogazici University, p.29.

[4] Baytaş, İ., 2020. Linear Regression Slide.CMPE 462 Machine Learning Bogazici University, p.13.