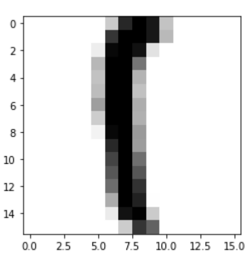
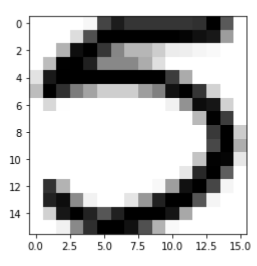
CMPE 462 - Project 1
Binary Classification with Logistic Regression
Due: April 23, 2020, 23:59

Task 1: Feature Extraction (35 Pts)

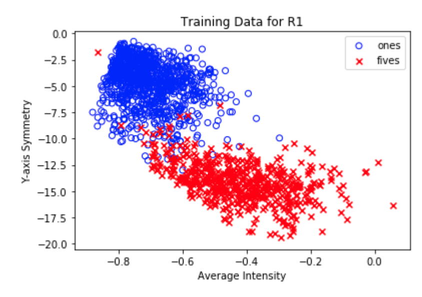
We visualised an example of "1" and "5" to begin the task.

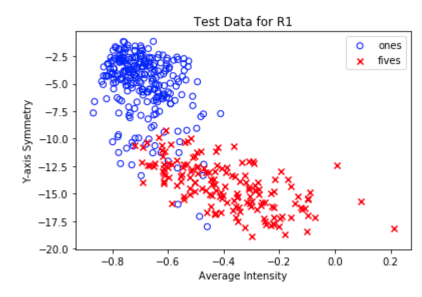




Implementing Representation 1: After visualising some data, we implemented representation 1 as instructed by the project guideline. It was a fairly straightforward process, we implemented symmetry calculation ourselves from scratch and computed the average intensity using numpy functions. We put those in a matrix.

Then we visualised the training and test data, below there are scatter plots for training and test data for representation 1.



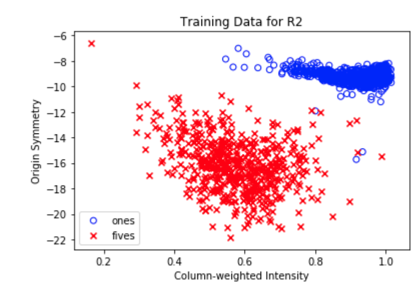


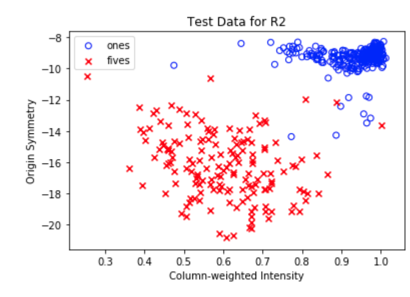
Implementing Representation 2: For representation2 we picked two features:

1. **Column-weighted Intensity**: Since digit 1 is more 'vertical' than digit 5, there are some particular columns in digit 1 that have much higher total intensity than in digit 5. Hence, intensity was calculated column-wise, in a way that increased the weight of these high-intensity columns. The range of pixel values were rescaled from [-1, 1] to [0, 1] and the sum of pixel values of each column was taken as a power of 10 (i.e. 10^{sum}). Then, after adding up all 16 columns this way, total value was taken to log10 and averaged out to reach the *column-weighted* intensity value for each data point.
2. **Origin Symmetry**: The negative of the norm of the difference between the image and its symmetrical with respect to the *origin* (center of the image) was calculated for each data point.

We calculated these features for each datapoint and we put them in a 2D matrix.

After calculating features for representation 2, we plottet them the same way we did for representation 1. Scatter plots are below:





Task 2: Logistic Regression

To implement the gradient descent of the logistic loss, we first derived the gradient of logistic loss function with respect to w.

Derivation of the gradient of the logistic loss with respect to w:

$$w(t+1) = w(t) + \eta\hat{v}$$

$$E(w) = \frac{1}{N}\sum_{n=1}^{N} ln(1 + e^{-y_n w^T x_n})$$

$$\frac{1}{1+e^{-s}} = \frac{e^s}{1+e^s}$$

Calculating ΔE using (1):

$$\Delta E = E(w(t+1)) - E(w(t)) = E(w(t) + \eta\hat{v}) - E(w(t))$$

Using Taylor series, we can approximate ΔE as:

$$\Delta E = -\eta \nabla E(w)^T \hat{v}$$

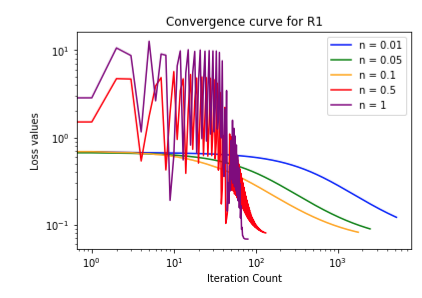
Calculating $\nabla E(w)$ using (2) and (3):

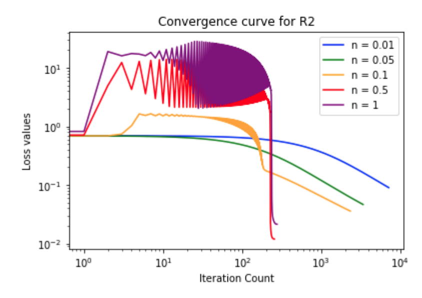
$$\nabla E(w) = \frac{1}{N}\sum_{n=1}^{N}(ln(1+e^{-y_n w^T x_n}))' = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{1+e^{-y_n w^T x_n}}(1+e^{-y_n w^T x_n})' = \frac{1}{N}\sum_{n=1}^{N}\frac{e^{-y_n w^T x_n}}{1+e^{-y_n w^T x_n}}(-y_n w^T x_n)'$$

$$\nabla E(w) = -\frac{1}{N}\sum_{n=1}^{N}\frac{y_n x_n}{1+e^{y_n w^T x_n}}$$

For the gradient descent algorithm, we calculated the loss function and updated the weight after calculating the gradient of the loss function with respect to the current weight. This algorithm was terminated when two difference between two consecutive loss values was below 10^{-5}.

We created lists for keeping the iteration count, loss values and weights. We tested our implementations for five different learning rates: 0.01, 0.05, 0.1, 0.5 and 1. We initialized w as a zero vector. Then using a for-loop, we ran the gradient descent algorithm for each learning rate, appending the iteration count, loss values and weights of each run to their respective lists. After that, we visualized the convergence curves.





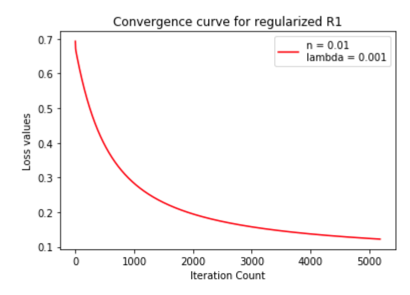
We also calculated the training and test accuracy for each learning rate for each representation.

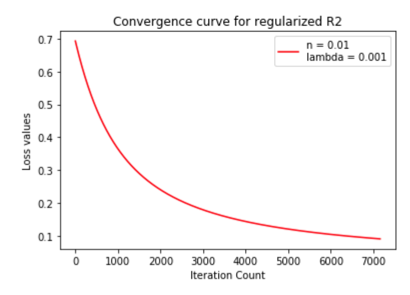
Eta value	R1 Training Accuracy	R1 Test Accuracy
0.01	97.50160153747598	95.04716981132076
0.05	97.56566303651505	94.81132075471697
0.1	97.69378603459322	95.04716981132076
0.5	97.50160153747598	95.04716981132076
1.0	97.82190903267136	95.28301886792453

Eta value	R2 Training Accuracy	R2 Test Accuracy
0.01	99.55156950672645	98.58490566037736
0.05	99.55156950672645	98.58490566037736
0.1	99.55156950672645	98.58490566037736
0.5	99.55156950672645	98.34905660377359
1.0	99.61563100576554	98.34905660377359

Logistic regression with ℓ_2 norm $||w||_2^2$ regularization

We implemented regularization into our algorithm by adding the necessary terms into the loss function and gradient calculations. We chose η as 0.01 and λ as 10^{-5} for this implementation. We again plotted the convergence curves and calculated training and test accuracy values.





R1 Regularized Training Accuracy: 97.50160153747598

R1 Regularized Test Accuracy: 95.04716981132076

R2 Regularized Training Accuracy: 99.55156950672645

R2 Regularized Test Accuracy: 98.58490566037736

5 fold Cross Validation

In order to find the optimal λ value for regularization, we implemented a 5-fold cross validation procedure by randomly splitting the data into five groups of equal size and cross-validating four of these groups with the fifth, held-out group for each possible combination. We fixed the randomization (seed(1)) so that the results do not change every time the code is run. We chose η as 0.01 and tried five different values for λ: 0.00001, 0.0001, 0.001, 0.01 and 0.1. Then, we calculated the mean and standard deviation values for each λ value, as shown in the following tables:

R1: Table

	Held-out gr.=0	Held-out gr.=1	Held-out gr.=2	Held-out gr.=3	Held-out gr.=4	Mean	Std
λ=1e-05	97.4358974	97.4358974	96.7948718	97.7564103	98.0769231	97.5	0.4252083
λ=1e-04	97.4358974	97.4358974	96.7948718	97.7564103	98.0769231	97.5	0.4252083
λ=1e-03	97.4358974	97.4358974	96.7948718	97.7564103	98.0769231	97.5	0.4252083
λ=1e-02	97.1153846	97.4358974	96.7948718	97.7564103	97.7564103	97.3717949	0.373779
λ=1e-01	97.1153846	95.1923077	96.474359	96.1538462	95.5128205	96.0897436	0.6844281

R2: Table

	Held-out gr.=0	Held-out gr.=1	Held-out gr.=2	Held-out gr.=3	Held-out gr.=4	Mean	Std
λ=1e-05	99.6794872	100.0	99.3589744	99.6794872	99.0384615	99.5512821	0.3268602
λ=1e-04	99.6794872	100.0	99.3589744	99.6794872	99.0384615	99.5512821	0.3268602
λ=1e-03	99.6794872	100.0	99.3589744	99.6794872	99.0384615	99.5512821	0.3268602
λ=1e-02	99.6794872	100.0	99.3589744	99.3589744	99.0384615	99.4871795	0.3268602
λ=1e-01	99.6794872	100.0	99.0384615	99.3589744	98.7179487	99.3589744	0.4532736

Task 3: Evaluation

After these steps we picked optimal η and λ values.

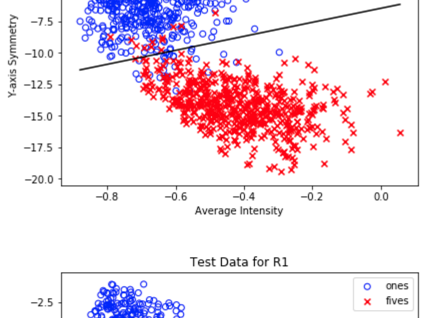
Best η = 0.01. Larger η values performed slightly better in terms of having the minimum loss values when the learning was terminated but their convergence curves were not smooth and were fluctuating a lot. We spoke to Prof. Baytaş about this and she advised against using fluctuating η values.

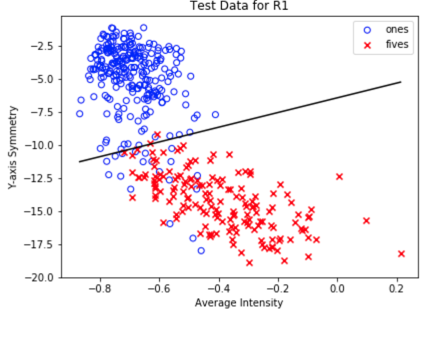
Best λ = 0.0001 Small λ values generally perform better than larger ones but as we go smaller there is not any difference between them. We picked the smallest λ as regularization does not seem to help.

Using these values, we implemented algorithms again. Here are the training and test accuracies with and without regularization:

	Training Accuracy	Test Accuracy
R1 w/o Regularization	97.50160153747598	95.04716981132076
R1 with Regularization	97.50160153747598	95.04716981132076
R2 w/o Regularization	99.55156950672645	98.58490566037736
R2 with Regularization	99.55156950672645	98.34905660377359

We visualized the decision boundaries by setting the hypothesis function equal to zero as follows.





To conclude the report, we would like to make these final remarks.

Regularization did not improve the generalization performance, because there was already very little difference between training and test accuracy, and changing lambda values did not effect accuracy values at all. This was the case for both representations. Since there were no improvements by regularization, we can conclude that we had no overfitting for either representation.

The features we came up with (Representation 2) separated the datapoints better (from %95 accuracy to %98.5).

To increase accuracy we would try the following: (1) try to add features that better separate the datapoints, (2) run pocket algorithm before implementing logistic regression to improve weights (this would also give us a head start and decrease iteration count), (3) decrease the 10^{-5} limit, the absolute difference between the current loss value and the loss value of the previous step that we used to terminate gradient descent further, (4) use neural network approach instead of logistic regression with gradient descent.