Ofer Orgal
300459898
oferorgal@mail.tau.ac.il

The files are in my directory on nova.cs.tau.ac.il:
/specific/a/home/cc/students/cs/oferorgal/ml/hw2/

If for some reason the files are not accessible, I created a share in my google drive to my files:
https://drive.google.com/drive/folders/0B6K9SrEqgeqRcXU2NHRLaDh5azA?usp=sharing

Folder content:
Python files:
- hw2.py - programing exercise 2

Reports:
- hw2.pdf - exercise report
- README

Plots and images:
- misclassified_0.png
- misclassified_8.png
- plot3a.png
- plot3b.png
- SGDWeights.png
- SVMWeights.png
- weights.png

How to use the files:

Each section in the exercise is accessible by running the file name and the section number and letter (i.e. 1a , 1b , ... 3a, ...)

programing exercise 2

Question 1:

sec. A:

How to use: ***python2.7 hw2.py 1a***

(I ORIGINALLY USES A TABLE CLASS IN PYTHON THAT DOEST EXIST ON NOVA SO I REMOVED IT I JUST PRINT THE DATA)

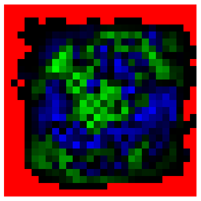
The *function: run_perceptron()* initializes a Table and the sample sizes to 5, 10, 50, 100, 500, 1000 and 5000 and run the function *run_perceptron_X_times()* for each sample size.
The *run_perceptron_X_times()* function train the algorithm (func: *train(data_norm, labels)*) and find the accuracy of it (func: *Perceptron_accuracy(data, labels, weights)*) according to the test_set.

The output is a table of the accuracies and 95, 5 percentiles of the different runs with the sample sizes.

sec. B:

How to use: ***python2.7 hw2.py 1b***

The function: *print_image("weights.png", train(train_data_norm, train_labels), 400)* gets the image name to output, the data (I use the *train()* function to get the weights) and 400 is a brightness value.
The output:



The colors represent:

- RED: Zero values of the weights in that area.
- GREEN: The weights values are positive.
- BLUE: The weights values are negative.

The greener the color is, the more weight is given to a pixel to determine it is 8, the bluer it is, the more weight is given to determine it is 0.

sec. C:

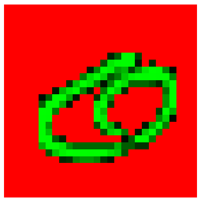
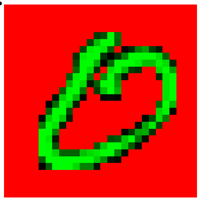
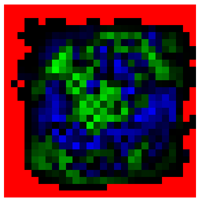
How to use: ***python2.7 hw2.py 1c***

I use the *Perceptron_accuracy(data, labels, weights)* with the *train(data_norm, labels)* function on the entire training set to find the algorithm accuracy on the test set.
The result I found is: 98.7717502559 % in determining if an image in the test set is 0 or 8.

sec. D:

How to use: ***python2.7 hw2.py 1d***

The *find_misclassified()* function returns 2 indexes of images from the test set (one with the label 0 and one with the label 8) that were misclassified by the algorithm.
The output:

Misclassified 0:  Misclassified 8:  Weights: 

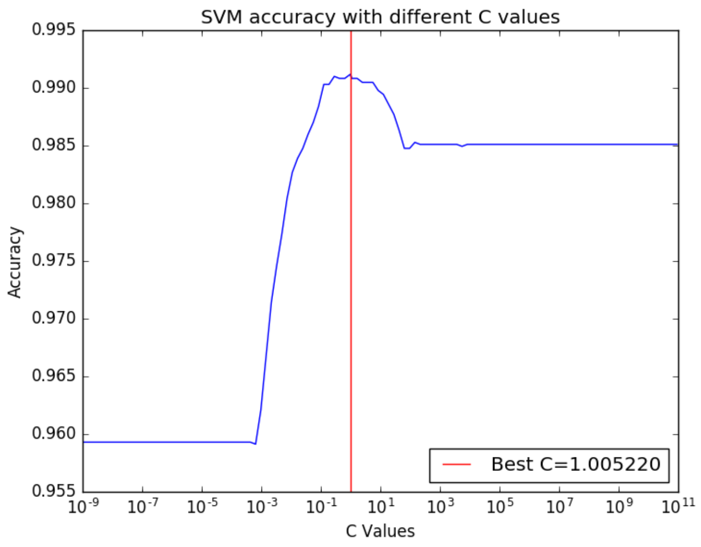
We can see that the misclassified 0 image was misclassified because there is a large portion of the image in a part where the weight image give a lot of weight to label it as 8.
The misclassified 8 image does not contain a lot of pixels in the area where 8 is should be (based on the weight image) and it is mostly in the area classified as 0.

Question 2:

sec. A:

How to use: ***python2.7 hw2.py 2a***

The function *SVM_find_best_C()* plots the accuracy of the algorithm for C values from 10^-10 to 10^10 and returns the C value of which the algorithm has the highest accuracy on the cross validation set.
We can see from the plot that the best C value is around 1.00522.
That is because I used normalized data (to length=1), if the data was not normalized, the C value was much lower.



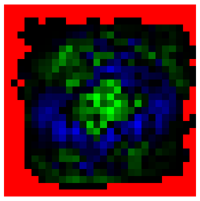
sec B:

The C value corresponds to the penalty error of the training algorithm. When the value is really low comparing to the data the accuracies is low but constant.
As we go higher, the accuracy improves up until the penalty is to high, then it decreases and stays at a constant value – the penalty value is higher then the data so it has no more effect on the algorithm.

sec. C:

How to use: ***python2.7 hw2.py 2c***

After I found the best C value for the algorithm, the function *SVM_weights()* print an image representing the weights trained be the algorithm.
Output:



We can see that the result is very similar to the image I got in the Perceptron algorithm (The color codes are the same).

sec. D:

How to use: ***python2.7 hw2.py 2d***

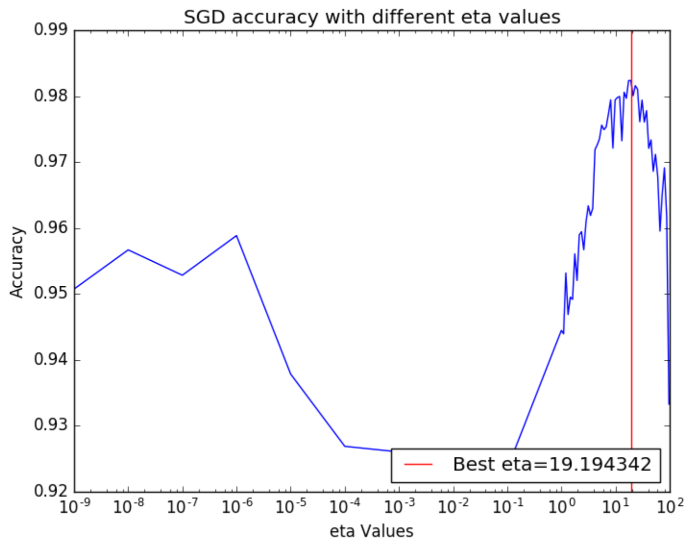
I use the LinearSVC score option the get the algorithm accuracy on the test set with the best C value I found previously.
The result I found is: 99.2835209826 % in determining if an image in the test set is 0 or 8.
We can see that the SVM algorithm preforms better then the Perceptron algorithm in this case.

Question 3:

sec. A:

How to use: ***python2.7 hw2.py 3a***

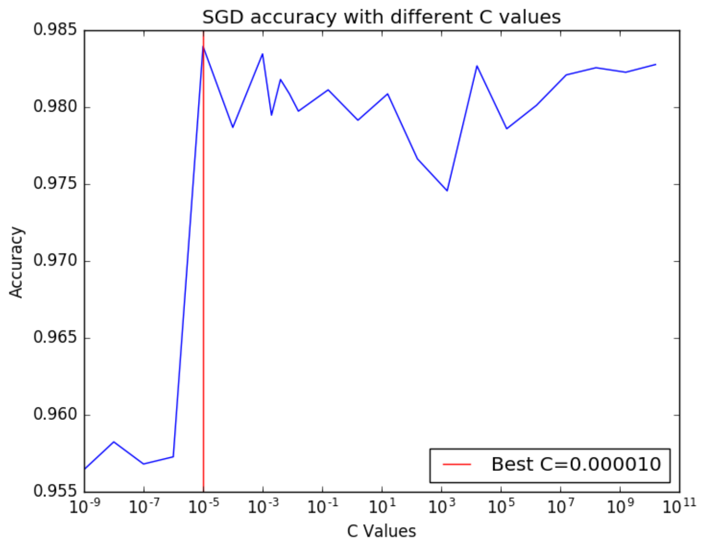
The function *SGD()* learn the best eta value for the schorske gradient dissent learning algorithm. For each eta value it draws 1000 random images from the training set and average the accuracy across 10 runs measured on the cross validation set.
The function outputs a plot of the eta values and the corresponding accuracies.
We can see that for a small value of eta (learning rate) we find a local optima of the accuracy but the eta value with the best accuracy is at around 20 (I found it at 19.194342) and the accuracy is: 98.34 %.



sec. B:

How to use: ***python2.7 hw2.py 3b***

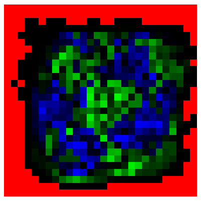
After I found the best eta value (with C value = 1), the function *SGD_find_best_C(best_eta)* is used to find the best C value. The function works the same as the one in section 3a but now the eta is constant and I change the C values.
The function plots the accuracy of the algorithm on the cross validation set as a function of the C values and returns the best C value (the C value varies across several values because of the algorithm uses random images each time).
The best accuracy is for C=0.00001 and it is: 98.39452%.



sec. C:

How to use: ***python2.7 hw2.py 3c***

The function *SGD_weights(best_C, best_eta)* get the best C and the best eta values found in the previous sections and print an image of the algorithm weights:



We can see that the result is similar to the images I got for the Perceptron and the SVM algorithms (color codes are the same).

sec. D:

How to use: ***python2.7 hw2.py 3d***

I use the SGD_accuracy(data, labels, w) with the *SGD_weights(best_C, best_eta)* function to train the algorithm and to find the algorithm accuracy on the test set.

The result I found is: 99.0276356192 % in determining if an image in the test set is 0 or 8.

Question 4:

I tried to use the SVM classifiers with Squared exponential kernel and Rational quadratic kernel and they showed better accuracy in determining if the image is 0 or 8 both on the cross validation set and on the test set.