ECE 236A Project 1 [Group 10]

Sunay Bhat, Samuel Gessow, Steven Zhiying Li, Terri Tsai, Dominic Yang

{sgessow, terri0716}@gmail.com

{sunaybhat,zhiyingli,domyang}@ucla.edu

Abstract

The purpose of the this project is to implement a robust linear program based classifier with the ability to classify handwritten digits from the MNIST data set online. The classifier should be able to scale up from a 2-digit classifier to a one vs. one classifier for all 10 digits. This report will detail our methods and results in attempting to get the best classification accuracy within project bounds.

Support Vector Method

We implemented the Support Vector Machine method with hard margins to train our binary classifiers. This method performs training by taking a set of points and with their corresponding known labels, and finding a hyperplane that best separates them. The data set will only contain two classes, so that ideally one class belongs to one side of the hyperplane, while the other class belongs to the other side. We do this by solving the following optimization problem, where x_i is each training data point, s_i is its correct label, and a and b are the parameters of the hyperplane we are looking for:

$$\min_{a,b} \quad \sum_{i=1}^{n} \max\{0, 1 - s_i(a^T x_i + b)\}$$

Suppose we are given N data points $x_1, \dots, x_N \in \mathbf{R}^n$ and their corresponding classes $s_1, \dots, s_N \in \{-1, 1\}$. We want to find a hyperplane $a^T x = b$, where $a \in \mathbf{R}^n, b \in \mathbf{R}$, which best separates the data points so that the points that are class $s_i = 1$ lie on one side of the hyperplane and satsify $a^T x + b \geq 0$, and points that are class $s_i = -1$ lie on the other side of the hyper plane and satisfy $a^T x + b < 0$. Some points may be outliers, laying closer with points in the opposite class, rather than with points with the same class label. There may exist no hyperplanes to take into account these outliers, so we create leeway for some points to cross over to the wrong side by using $1 - s_i(a^T x_i + b)$ as the positive value penalty associated with being on the wrong side of the hyperplane. While some training points inevitably have to be separated onto the wrong side, we minimize that as much as possible by minimizing the penalties of all such points. This is what we used specifically for our project. This problem can be written as a linear program using standard transformations:

$$\begin{array}{ll} \min\limits_{t,a,b} & \sum\limits_{i=1}^{N} t_i \\ \text{s.t.} & 0 \leqslant t_i, i = 1, \dots, N \\ & 1 - s_i\left(a^T x_i + b\right) \leqslant t_i, i = 1, \dots, N \end{array}$$

In the multi-class case, where a test point could one of n classes, like one of ten digits, $\frac{n(n-1)}{2}$ binary trained classifiers will be applied to the test point, and the most common labeled assigned to the test point out of the $\frac{n(n-1)}{2}$ assignments will be our final classification of the test point.

Optimizing the Solver

We noticed that for large data sets the solver would error out and take a really long time to solve. We then tried increasing the tolerances for two of the solvers that gave us the best results: SCS and ECOS. We proceeded with the ECOS solver for its increased reliability and systematically decreased the tolerances while looking at the overall classification accuracy until we settled on our final settings.

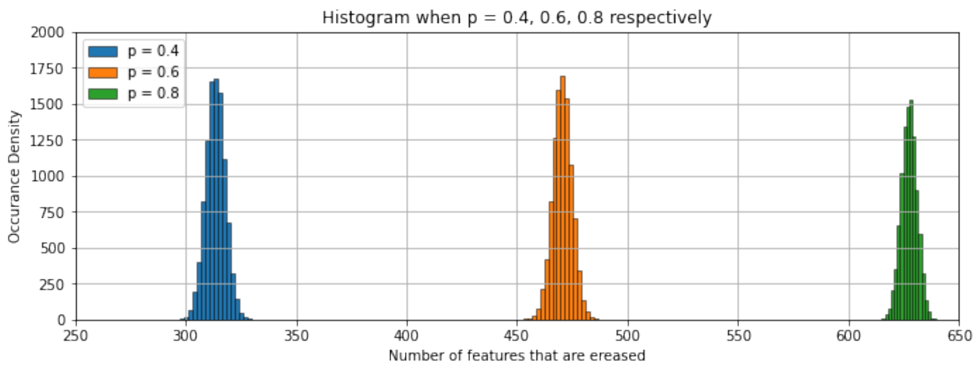

Figure 1: Histogram of number of features deleted for $p = 0.4, 0.6$, and 0.8

Test Data Corruption

As we can see in Figure 1, these three clusters of histogram represents the frequency of removed pixels when given $p = 0.4, 0.6$, and 0.8, respectively. The horizontal line represents the number of removed pixels, ranging from 0 to 726. These are data aggregated from 10 test trials for each p value.

Based on the central limit theorem, we can see the distribution of pixel corruption tends to the Gaussian distribution. These three distribution are centered at the expected value $784 \times p$, which are about $314, 470$, and 627, respectively.

Training Method

After optimizing the linear program and solver inputs, we focused on improving the accuracy of our classifier by modifying the way in which it was trained understanding the corruption scheme. First, we wanted to compare training with corrupted data in two different ways. The first would be corrupting the data using various p values or corruption probabilities. Additionally, any N sets of training data could be made by simply randomly corrupting a dataset again. Thus, the first plot in Figure 2 shows test data accuracy results after corrupting the training data with $p = 0.2, 0.4$, and 0.6, and then doing so with two sets of each. It is immediately clear that corrupting training data vastly improves the classifier performance, and the higher 0.6 corruption level offers the best results at all levels of uncorrupted and corrupted data (note 0.8 corruption level not plotted here). The payoff of using 2 sets of corrupted data was very marginal and not immediately clear. A similar analysis on the full 10 class classifier showed similar results for training with corrupted data is also shown in Figure 2.

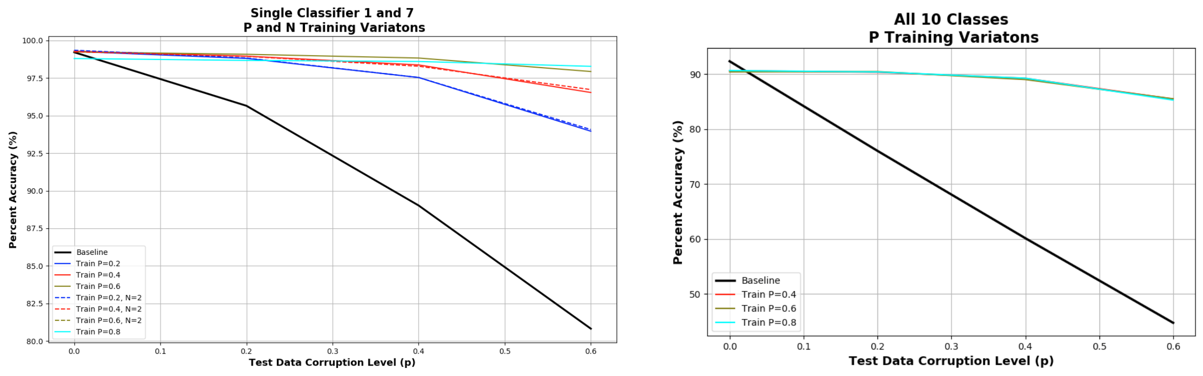

Figure 2: Classification accuracy on digits 1 versus 7 and on all digits when training at different corruption levels and different amounts of corrupted data (for 1 and 7 only).

In order to further investigate the benefit of multiple sets of training data per classifier, we ran a scheme in which we averaged the results of multiple trained classifiers with 0.6 corrupted training data to determine if accuracy could be improved that way. This showed noticeable accuracy gains by utilizing more sets of data and averaging the results. The results can be seen in Figure 3.

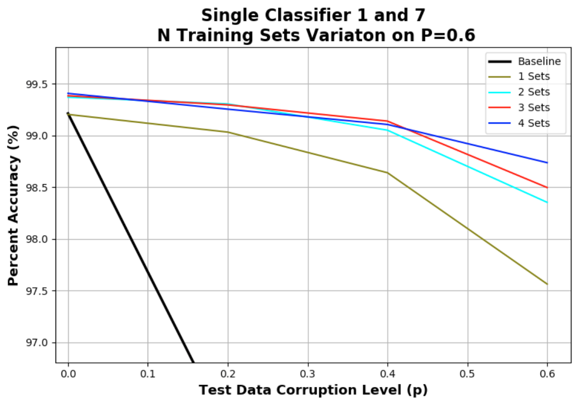

Figure 3: Classification accuracy on digits 1 versus 7 when training on with multiple sets of p=0.6 corrupted data

The final method we investigated was utilizing a 'tiled' method, in which we triple the size of the training data set having one third corrupted at 40%, one third at 60%, and one third at 80%. Although the computational time increases more than linearly (unlike the multiple sets averaged which scales linearly), the classification accuracy improvements were significant at higher test corruption levels as compared to a 0.6 corrupted average of 3 weights classifier. This can be seen in the plots in Figure 4.

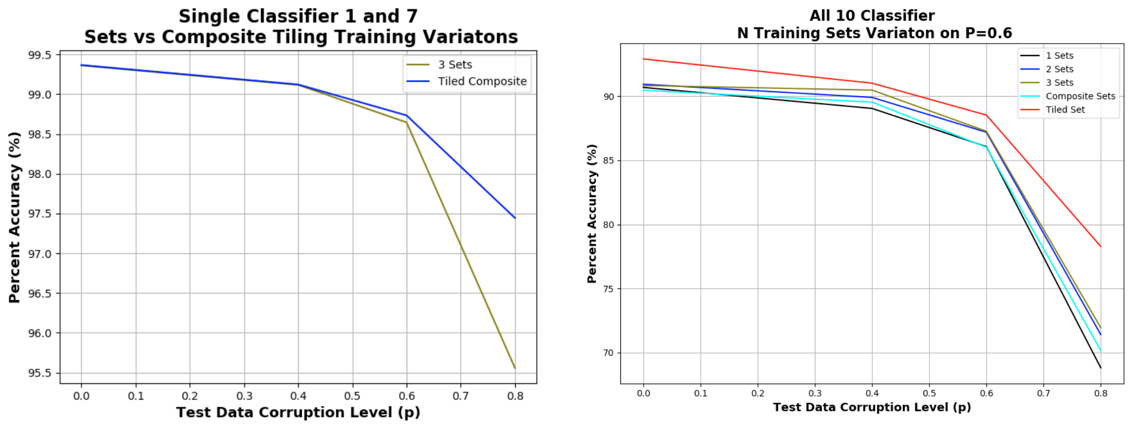

Figure 4: Classification accuracy on digits 1 versus 7 and all digits when training set has multiple versions of corrupted data for fixed p and for varying p.

After verifying these gains translated to the generic 10 class classifier, this 'tiling' of multiple corruption levels of training data was the method we chose to implement due to the increased accuracy performance across corruption levels. Note that the 'tiling' varies with the p-value that is passed in at training, so that our training data is corrupted at p-0.2, p, and p+0.2.

Final Results

For digit 1 and 7, the average percentage accuracy of correct classification after running the test 10 times is: [99.4 99.18 98.8 97.42] for p=0, 0.4, 0.6, 0.8 respectively.

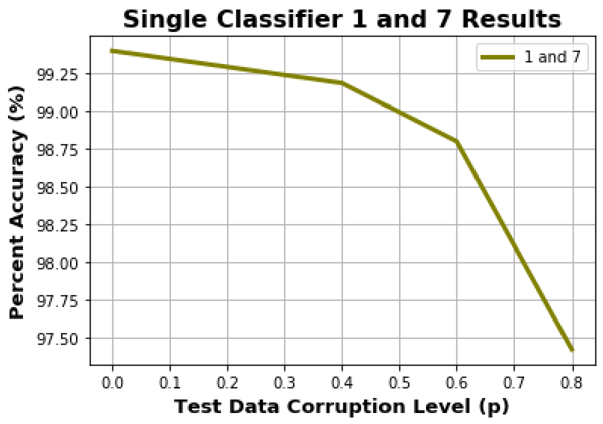

Figure 5: Average percentage accuracy of correct classification after running the test 10 times