Reproducing Results of Guassian Kernel SVM classifers on the MNIST Dataset

Nolan Hartwick
University of California San Diego
nhartwic@ucsd.edu

ABSTRACT

The MNIST database is a widely known, well studied dataset of hand written digits. A wide variety of machine learning strategies have been applied to this data set with varying results. The website dedicated to the MNIST database details some of these results, including exactly one result without any citation or reference. It is reported that an SVM with a Gaussian kernel is capable of achieving a 1.4% error rate on the MNIST data set with no image prepossessing. This essay details my attempts to replicate these results, and my exploration of the application of Gaussian SVM classifiers to the MNIST dataset. I was unable to perfectly replicate the reported results on unprocessed images but did achieve a best error rate of 1.47%.

Keywords

Machine Learning; Support Vector Machine (SVM); MNIST;

1. THE MNIST DATASET

The MNIST database contains black and white images of hand written digits of size 28x28 . It is a subset of a much larger NIST database. Each character in each image has been algorithmically centered and has been labelled by a human according to what character the image displays. These images are split into two different subsets, a ’Training’ set containing 60,000 images and a ’Testing’ set containing 10,000 images. The typical machine learning task with the MNIST dataset is to design a classifier capable of classifying images according to which digit is in the image. The best accuracy, 0.23% test error, listed on the MNIST website was accomplished using a committee of 35 convolutional nets. There are many other reported results including exactly one result that is totally unreferenced and unsupported. It is claimed that an SVM with a Gaussian kernel was able to achieve a test error rate of 1.47% with no preprocessing. As the only unsupported result, I thought that I would attempt to recreate these results and offer my work as reference to the MNIST managers.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Digit	Testing	Training
0	0.098	0.0987
1	0.1135	0.1123
2	0.1032	0.0993
3	0.101	0.1021
4	0.0982	0.0973
5	0.0892	0.0903
6	0.0958	0.0986
7	0.1028	0.1044
8	0.0974	0.0975
9	0.1009	0.0991

Table 1: The contents of the training and testing sets reported as frequencies of each digit.

Figure 1 contains some examples of the images found in the mnist data set. Some basic statistics covering the contents of each of the data sets can be found in Table 1. Figure 2 displays the ’average’ representation of each character in MNIST. In order to generate the ’average’, each image is first transformed by z-scoring each pixel in that image relative the mean and standard deviation of the pixels in that image. This z-scoring methodology acts to normalize the images by brightness. These z-scored averages are then mapped back onto the discrete 0-255 range of pixels.

The results displayed in Table 1 demonstrate that their aren’t any apparently significant changes in the contents of the two data sets. For each digit in the datasets, the frequency of that digit in each dataset is similar. Each digit occupies approximately 10% of the total data. If there were any large biases in the images themselves, the bias should have been visible in Figure 2. As each average image is similar between the training and the testing set, the images themselves are similar. Neither of these tests are particularly rigorous, but if there were any huge problems with the data, we should have detected it. That many others have been able to easily use this data suggest that the data sets are good as well.

Once I had established that the MNIST dataset was apparently good, I elected to split the ’Training’ set into a smaller training set $T1$ by randomly selecting 50,000 random images from the original training set. The remaining 10,000 images from the original training set were then used to form a validation set, $V1$. The $T1$ set was used for training SVM classifiers with various hyper parameters. Each of these classifiers were then testing on the the $V1$ set in order to identify good hyper parameters.

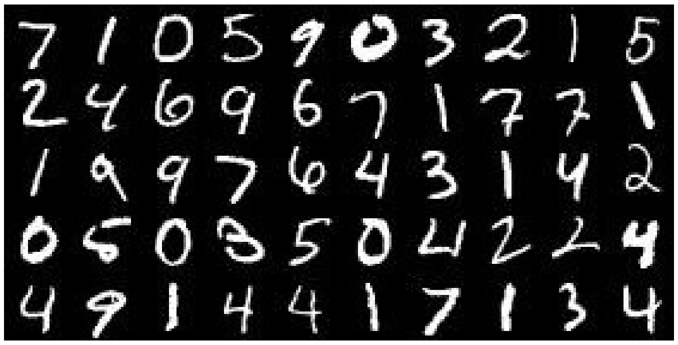

Figure 1: A subset of the MNIST testing dataset meant to display some examples of the hand written characters in the dataset.

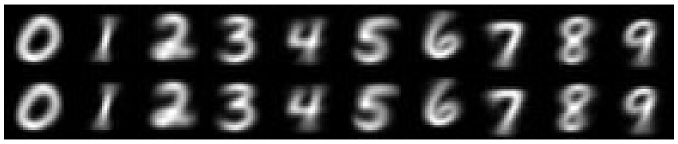

Figure 2: A representation of the data in MNIST generated by averaging images that shared the same label. The top row contains average testing images while the bottom row contains average training images.

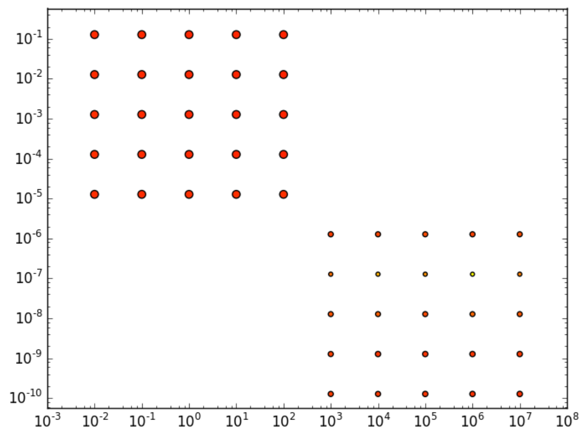

Figure 3: A log scale graph displaying, for each tested combination of hyper parameters, the performance of the resultant model measured as the error rate on the $V1$ set. Each dot is colored such that the more red a dot is, the larger the error was. Each dot is also sized such that the larger the dot is, the larger the error rate was.

2. DEVELOPING THE MODEL

A Support Vector Machine (SVM) is a useful machine learning tool that can be used for both regression and classification tasks. An SVM works by mapping the input data points into a high dimensional space, and then separating these input using a hyperplane that maximizes the distance between the hyperplane and the categories that it seeks to separate as well as minimizing the number of misclassified examples. Many different types of mappings are possible. Because the result I wish to replicate used a Gaussian (rbf) kernel, I will do the same. The rbf kernel requires the specification of a hyper parameter, *gamma*, that partially controls the behavior of the kernel function. The only other important hyper parameter is the penalty parameter, C, which controls how large the penalty for misclassified examples are during training.

Identifying good parameters was the real challenge to creating a good predictor. The SVM implementation that I used for my experiments was the python sklearn svm. Because training on the full $T1$ dataset was prohibitively time consuming, taking as much as a few hours when using bad hyper parameters, I elected to generate a random subset of $T1$, that contained 10,000 images and used this set for training models during exploration of hyper parameters. I bench marked the performance of the tested hyper parameters by comparing the resultant SVM classifier's performance on the $V1$ set. The primary benchmark used was Error rate, defined as the number of incorrect classifications divided by the total number of classifications. I explored the hyper paramters using iterative combinations of grid search and gradient descent.

Because the default parameters for the sklearn SVM were $gamma = 1/len(features)$ and $C = 1$, I began by exploring the space immediately surrounding these defaults. I built a small log scale grid and explored the space within a multiplicative factor of 100 to 0.01 of the defaults. Because the best of these parameters was at the periphery of the results, I shifted my window to move down the error gradient and repeated the experiment. During this second test, I looked at values of C ranging from 10^3 to 10^7 times the default and values of gamma ranging from 10^{-3} to 10^{-7} times the default values. The best test was within the grid boundaries with values of $C = 10^6$ and $gamma = 1/len(features) * 10^{-4}$. Figure 3 graphs the error and associated hyper parameters for these tests.

Because the changes in the C parameter had little impact on performance, I opted to fine tune my search only on the *gamma* parameter. I tested values of *gamma* between 10^{-3} and 10^-5 times the default using a logarithmic step size of 0.1. The best test value occurred at $gamma = 1/len(features) * 10^{-3.5}$. The final step to perform was to let the SVM train on the full $T1$ training set using the good parameters I had discovered, and measure its accuracy on the $V1$ set and the MNIST 'Testing' set. After training with parameters $C = 10^6$ and $gamma = 1/len(features) * 10^{-3.5}$ on the full dataset, the svm had achieved a $T1$ set error of 0.0%, a $V1$ set error of 1.4%, and a 'Testing' set error of 1.47%. Table 2 displays the ability of the fully trained SVM to classify each type of character. '9's were the most difficult digit to classify and had the highest error rate. In total 19% of the errors made by the SVM were the result of classifying images labelled as being nines incorrectly.

3. DISCUSSION

I was able to get very close to the claimed result of 1.4% error rate. The real work in doing so was exploring the hyper parameters. Initial tests on unprocessed images were very disappointing. Training took multiple hours and resulted in

Digit	Error
0	0.0061
1	0.0052
2	0.0164
3	0.0148
4	0.0132
5	0.0168
6	0.0104
7	0.0184
8	0.0184
9	0.0277

Table 2: For each digit K in the testing set, the ratio of those digits that are labelled K but are misclassified by the SVM as not K to the total number of Images labelled K. In other words, the error rate of the fully trained SVM for each digit.

apparently huge over fitting. With default parameters on the full $T1$ data set, the SVM produced a training set error of 0.0% and a $V1$ set error of 89%. Initial exploration of the hyper parameter space didn't result in much improvement either. The performance of the SVM on the data set proved to be very sensitive to changes in the *gamma* parameter and less sensitive to changes in the C parameter. Increasing or decreasing *gamma* by a factor of 10 from the optimal values resulted in a doubling of the $V1$ set error rate. By comparison, increasing or decreasing C by a factor of 10 from the optimal value resulted in an increase in the error rate by a factor of less than 1.10. Further iterations of testing hyper paramters would likely result in slightly improved performance, but also risks over fitting on the validation set. Even with only the Log scale exploration that I performed, I had begun to see some slight over fitting with a final $V1$ set error of 1.36%, compared with the final 'Testing' set error of 1.47%. In any case, my results seem to indicate that a 1.4% test error rate is achievable using only a Gaussian kernel SVM with no preprocessing needed. I hope that my results will lend credence to the unsupported claims found on the MNIST website.