ECE 532: Final Report
Michael Giordano

Introduction

I used the MNIST dataset for my project. The goal of the dataset is to classify handwritten digits 0-9, presented in a 28x28 image. The dataset consists of 60,000 training examples and 10,000 test examples. I decided to write and evaluate three different algorithms for this classification task. Identical pre-processing in the form of PCA was applied to the dataset before classification. Algorithms explored include a linear classifier, k-nearest neighbor model, and a neural network with one hidden layer. Training across different data splits was performed on the linear classifier and neural network. The k-nearest neighbor model was excluded from this as a result of it's long run-time.

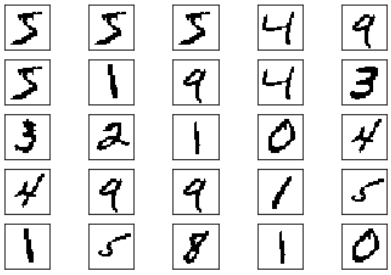

Figure 1: Examples of Handwritten MNIST Digits

All code, process, and results have been congregated and cleaned up and can be found at this github page under the notebook "Final_Results".

Preprocessing

Data Splits

The MNIST data set was split by makers into training and test groups based on sub-grouping within the data. Specifically, some of the data comes from employees at the Census Bureau while others come from high school students [1]. The training-test split provided accounts for this difference and makes sure both groups are represented in the training and test data to the same degree. To explore the impact of this decision, the 70,000 data points were randomly shuffled in 7 different sets of 60,000 training examples and 10,000 testing examples. A comparison of these random splits and the splits provided by the makers is included in the analysis.

Principal Component Analysis (PCA)

Principal Component Analysis was used to create a low-rank approximation of the training data. The 60,000, 28x28 data points were vectorized into 784x1, means were removed and then the data was deconstructed using the singular value decomposition into matrices U, s, and V^T. I then plotted the log of the singular values in order. The graph (see below) showed a clear drop-off when s falls below one (equivalently log(s) falls below 0).

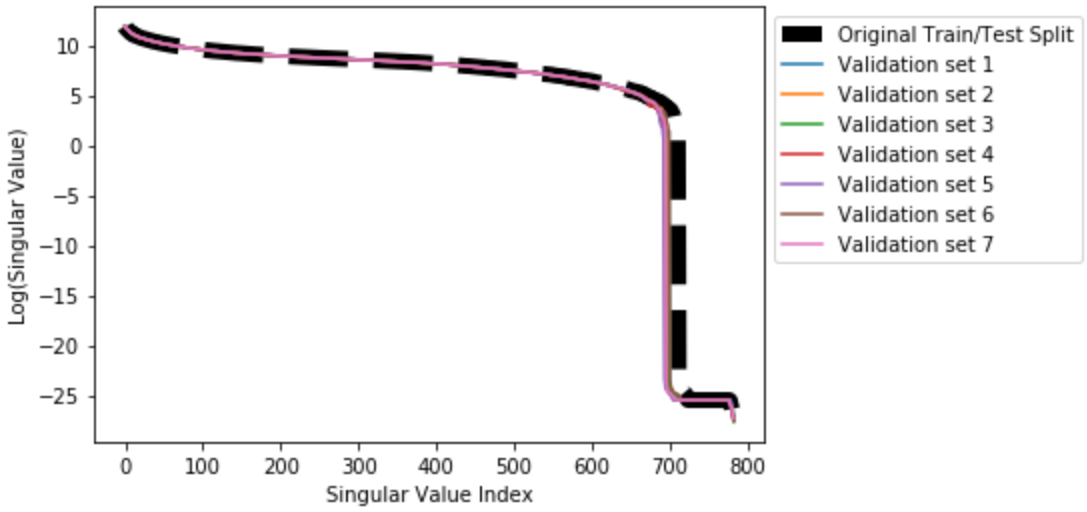

Figure 2: Graph of the Log of the Singular Values by Index. The original test split, as well as the other splits, fell off around index 700

Only singular values of s > 1 were kept. Columns of U and rows of V^T with index greater than the singular value kept were then dropped. The low-rank approximation was then reconstructed with these and adding back the means.

$$A_{low-rank\ approximation} = (U_{:,:712} * s_{:712} * V^T{}_{:712,:}) + \mu$$

Equation 1: Low-Rank Approximation Reconstruction Example where 712 singular values are kept. μ is the mean added back

Results

Linear Classifier

The first algorithm I investigated was linear classifier. In this case, 10 classification boundaries are created, one for each label. For instance, the classification boundary for seven determines whether something is or isn't a seven.

To train the least squares model, 10 different sets of labels were created, corresponding to whether something was or wasn't a number. Then, the below regularized least-squares equation was applied to the low-rank approximation data to come up with weights.

$$w = (A^T A + \lambda I)^{-1} A^T d$$

Equation 2: Regularized Least Squares Equation

Weights were applied to the test data with $\lambda = 10^{-5}$ and error results for each digit, in each split are shown below. Eg. Error rate of .04 for digit 3 means 4% of 3's were on the wrong side of the three/not a three boundary. Clearly, the test splits don't make a significant difference.

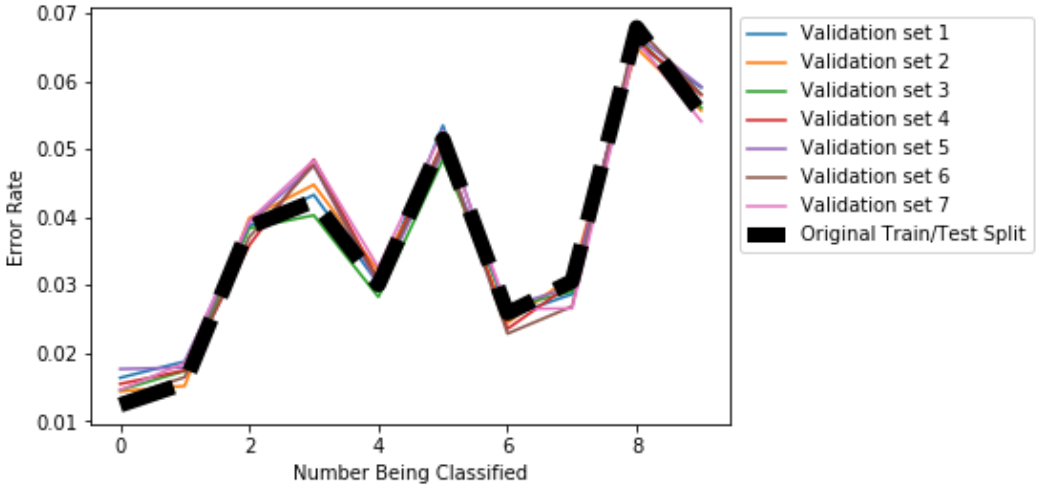

Figure 3: Error Rates Across Digits in Linear Classifier

I then labelled each image with a number by taking the maximum output of the ten classifiers, ie. which digit is this "most like". The resulting error rate for the original split was .1466 and the average error rate for the other set was .1547.

Lambda values 10^{-7} through 10^{9} were tested on the original split and are shown in the table below. Error rates did not change significantly unless a very large lambda was used. Much of the noise in the data was most likely eliminated in the low-rank approximation, leading to lambda values not having a great impact.

λ	10^{7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{4}	10^{5}	10^{6}	10^{7}	10^{8}	10^{9}
Error	.1466	.1466	.1466	.1466	.1466	.1466	.1466	.1466	.1466	.1467	.1466	.1466	.1452	.1435	.1424	.1547	.1959

Table 1: Error Values for Linear Classifier across lambda values

K-Nearest Neighbor Classifier (KNN)

To classify a new instance, KNN simply takes the most common class of the k training examples closest to it in terms of Euclidean distance squared (see below). Ties were broken in an arbitrary fashion. The squared euclidean distance was used to save computation time as ranking the nearest examples by euclidean distance would result in the same outcome.

$$distance = \sum_{i=1}^{28} \sum_{j=1}^{28} (x_{ij} - y_{ij})^2$$

Equation 3: Euclidean distance equation for k-nearest neighbor algorithm

Five values of k were tested on the original data split with the error results given below.

k	1	5	9	13	27
Error	.0309	.0309	.0338	.0347	.0396

Table 2: Error Rates for K-nearest Neighbor for various values of K

Neural Network

A two-layer neural network using keras was built for classification. The hidden layer used the relu activation function. The relu function was chosen in this layer to make training less time consuming because negative values are zeroed out and this layer is much more dense. The softmax function was chosen as the output layer (with ten nodes) to provide a probabilistic interpretation (a value between 0 and 1) of which output the example was. Sparse, categorical, cross entropy was chosen as the measurement of loss since it avoids outliers pulling the data as much (compared to mean squared error) and allows for non-one-hot encoded examples to be entered. The default learning rate of .001 was used and not changed since the algorithm converged in a reasonable amount of time.

Different numbers of nodes in the hidden layer were tested on the original test split. Between 100 and 900 nodes were tested, with around 600-800 nodes producing the best results. After further analysis, the number of singular values used in the SVD was used as the number of hidden nodes (see notebook for more details). This network was trained and tested for all of the cross-validation sets and the original train/test split and results are shown below. The error rate for the original split was .0462 and the average error rate for the cross-validation splits was .03487.

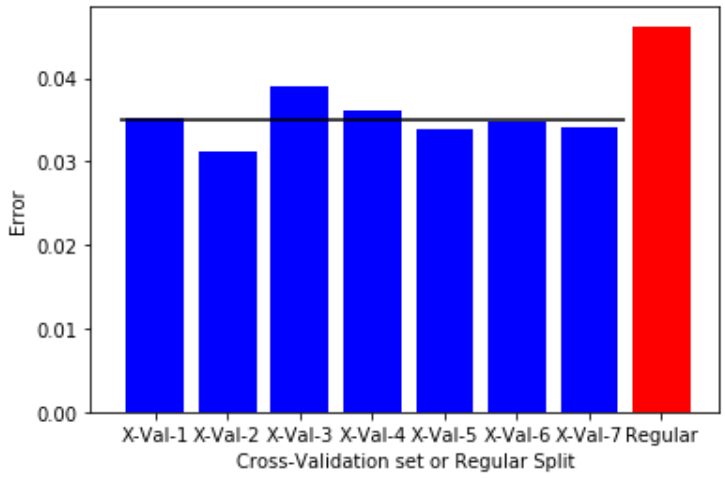

Figure 4: Error rates for Neural Network in regular and other data splits. Note: A new classifier was trained for each of these splits to test the value of the authors deliberate splitting of the data

Analysis

The k-nearest neighbor classifier surprisingly produced the best error rates of any of the classifiers with an error rate of ~3%, although the neural network was close behind with error rates ~3.5%. The linear classifier didn't perform nearly as well but still came up with a reasonable error rate of ~14.5%. Clearly, the data is clustered fairly nicely and doesn't contain a ton of noise since the k-nearest neighbor method can perform well. This is supported by the fact that ~700/784 possible singular values are significant. Ie. Out of the 784 dimensions in the training example, ~700 include at least somewhat relevant features.

The strength of k-nearest neighbors is its simplicity and ability to classify easy to classify data. Since our data is fairly clean, it performs well on the dataset. However, if more complicated/noisy data were to appear, it's ability to classify would most likely deteriorate. In addition, the calculation of distance is time consuming and needs to be computed across all 60,000 training examples for each test example.

The linear classifier is also relatively simple to write and has a closed-form solution. In addition, it is relatively fast for data of this size. However, this might become slower as data grows since it requires a matrix inversion. Furthermore, the inability to represent complex decision boundaries caused the error of this classifier to be higher than the rest.

The neural network is good at being able to model complex decision boundaries and produces a relatively low error. However, error for this method isn't guaranteed to converge to a global minima and can make deciding training parameters difficult. It was interesting to see the lowest error rate on the system seem to hover around the amount of significant singular values kept. This makes sense as training of the neural network then might converge to a solution where each hidden node corresponds to a particular feature.

One of the more interesting parts of this project was comparing the effects of splitting the training/test data in a random, unorganized way. The non-uniform splits produced a slightly higher error for the linear classifier (about one percent). This is exactly what I would expect as the classifier wouldn't necessarily have the correct splits of training data to learn. However, the error for the neural network improved by ~one percent when split this way. This is not what I would expect and could be just due to the algorithm converging to a better solution by chance in the new splits.

Conclusion

PCA is helpful pre-processing to produce usable results on this dataset. The k-nearest neighbor and neural network algorithms both performed well with similar error rates. However, the neural network will most likely be better suited for data which becomes noisy and should be preferred. The linear classifier produced reasonable error but is not the optimum classifier for this dataset due to limitations caused by non-complex decisions boundaries. It's unclear whether deliberate stratification of the data into training/test sets done by the datasets makers is significantly different from random sampling.

References

[1] Y. LeCun, “THE MNIST DATABASE,” *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 11-Dec-2020].