Brenden Hein

CSE 802

Dr. Ross

MNIST Handwritten Digit Classifier

1. Introduction

Computers have spent countless hours attempting to recognize something that is handwritten into something it is able to understand. This comes in many forms, ranging from letters of the alphabet to punctuation and sentence structure. Another example is taking numbers, symbols, and mathematical operators, turning them into equations. In this project, the question of how computers can recognize handwritten digits will be analyzed. Handwritten digits are still an integral part of life today, so a computer's ability to classify them can be significant. Applications that are fed an image of a handwritten equation, can interpret the image, and solve it is one such example of how a handwritten digit classifier can be beneficial. Online banking and check depositing is another area that where a handwritten digit classifier can be essential. Since a well-developed handwritten classifier can play an important role in many facets of computer science and life, this project will attempt to develop a simple, effective handwritten digit classifier, comparing it to some of the current standards of today.

The biggest issue presented with handwritten digits is the number of features that can potentially be extracted from an image of the digit. The MNIST dataset, which will be used in this project and discussed in more detail later, uses 784 features to describe each pixel of the image. That number of features can be computationally exhaustive to a computer and can very quickly lead to overfitting the model to the training data. This issue leads to the first question: how can the model significantly reduce the number of features without losing pertinent information about the pattern. Another problem with handwritten digit classifiers is that the form of the data given a subset of features is unknown. Thus, non-parametric estimation methods, such as K-NN and Parzen windows must be used, even though they can be memory-intensive.

Many of today's current handwritten digit classifiers are extremely effective, but at the same time, they can be complex. One such example involves convolutional neural networks, which is extremely accurate, making it a near perfect option for classifying handwritten digits using the MNIST dataset. Another example involves using support vector machines to classify digits; this method is also very successful when it comes to classification accuracy. In this project, however, a new, simpler classifier will be developed with a unique set of features, and it will use non-parametric classification methods. It will then be compared against the current standard to see if the level of complexity yields substantially better results, or if a simpler model can still achieve similarly effective results.

2. Project Plans

The first part of this project will involve preprocessing and feature extraction. The MNIST data set provides 60,000 training samples and 10,000 testing samples, with each sample being a handwritten digit: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Each of these samples contain 784 features, which when all used together for classification, can lead to poor results. Therefore, a subset of features will be gathered. The set of features derived for this project involves a 28 x 28 bounding box surrounding each of the samples. First, the samples will be reshaped from a 1 x 784 feature vector into a 28 x 28 matrix, with each of the ith-jth indices corresponding to the image's ith-jth pixel. Each pixel is also rounded up to 1 if its 1-byte color value is greater than the 0, a process called binarization.

After, the images will be de-skewed. Many people, when they naturally write, will do so on an angle. Given that the features extracted will be based around distances in a bounding box, this slanting can cause a problem. Thus, an affine transform will be applied to every image, so characters in the MNIST dataset will be rotated to an upright orientation if they are off-angled. This will hopefully eliminate some unnecessary intraclass variability due to characters being angled. The code used to de-skew the images in this project can be found in [3].

Once the preprocessing is complete, features can be extracted. The set of features that will be extracted will be based on the distances from the corners and the midpoints of the bounding box surrounding the image to the handwritten digit. In this process, the algorithm works inward from the corners and midpoints, stopping when it encounters a pixel of the handwritten digit. Therefore, a total of 8 features will be extracted, defined as follows: the distance to the digit from the…

1. Upper Left Boundary
2. Lower Right Boundary
3. Upper Right Boundary
4. Lower Left Boundary
5. Middle Left Boundary
6. Middle Right Boundary
7. Middle Upper Boundary
8. Middle Lower Boundary

Histograms of the individual features given all classes will be generated, which can give an idea of the separability of each class for a given feature. This can provide initial insights on what features may have high interclass similarity or low intraclass similarity, and oppositely, high intraclass similarity or low interclass similarity.

As another option to explore, as eight features can be visually and computationally difficult to work with, methods such as using MDA or PCA could be used to transform feature sets into lower dimensional space. In this case, MDA may be used on the 8-dimensional bounding box feature vector to help visualize the separation of points in 2-dimensional or 3-dimensional space. While using MDA may not significantly reduce the computational complexity of K-NN and Parzen windows for lower-dimensional feature vectors, when using something such as the entire original set of features, MDA can be beneficial, although the loss of information may potentially be too significant to qualify it as a good classifier if the dimensionality is reduced too much. However, the potential benefits of using a feature transformation method such as MDA make it a viable option to test out during this project.

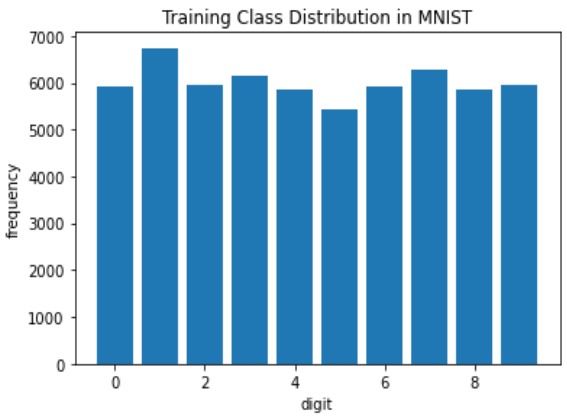
In terms of training and testing patterns, as these datasets are relatively large, they can be computationally demanding for algorithms such as K-NN and Parzen windows, as for every test point, all training patterns are looked at. One option for this may be to investigate potential complexity reduction algorithms for K-NN or Parzen windows to shorten its runtime. Another potential option is to take multiple subsets of the datasets to evaluate the classifiers effectiveness. Not only would this substantially decrease the runtime of testing these patterns, but it would also provide multiple error rates, which would allow for the calculation of the mean and variance of these error rates, along with visual data descriptions.

When it comes to the results, there are a few important things to look at. The most important is the error rate, which defines the classification accuracy of the model. While it may just be represented as a single data point, by partitioning the dataset into smaller subsets and generating multiple error rates, a mean error rate along with its spread can be derived, detailing the accuracy of the model better. The difference in error rates based on the number of training points, the ideal window size for Parzen windows, or the ideal number of K nearest neighbors can be analyzed, as well. These can lead to the findings such as if the if Parzen windows or K-NN works better, and what is the optimal window or K nearest neighbor size. As well as the error rate, confusion matrices can be generated to show exactly which classes had better or worse classification accuracy, along with how significant it was.

The effectiveness of the classifier will also be compared against the most accurate and the current standard of MNIST classifiers: convolutional neural networks. While it is unlikely that this classifier will be as effective or well developed as something such as CNN classifiers, the goal of this project is to see if using a simpler, non-parametric, distance-based classifier can yield results that are not too dissimilar to the current standard of handwritten digit classifiers.

3. MNIST Dataset

As stated early, this project will use the MNIST handwritten digit dataset. This dataset was downloaded from [1], where two csv files were provided: a dataset containing the 60,000 training samples and a dataset containing 10,000 testing samples. Each sample in the downloaded dataset is a 784-dimensional feature vector for a handwritten digit (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). The number of samples for each class is relatively uniform, as seen below.



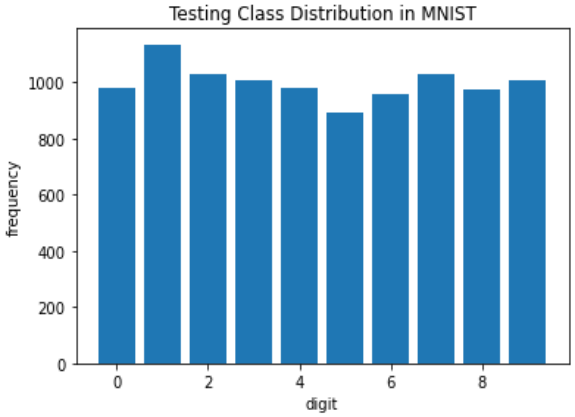


Fig. 1. and Fig. 2. The frequency distribution of the 10 classes for the training/testing datasets.

For each sample in the original dataset, there are also 784 features, with each feature being represented by an ith-row and jth-column pixel, where a single byte is used to represent the pixel's color. Every sample is represented as a grayscale image. The 784 features are arranged in a 28 x 28 image. An example of a pattern representing the digit, 0, is seen in figure 3.

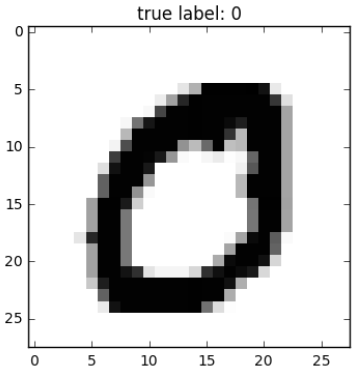


Fig. 3. An example pattern of the digit, 0, in the MNIST dataset, gathered from [2].

For this project, the 2 csv files were used. In both the training file and testing file, an image was represented by a single row, filled with 785 columns. The first column of every row represented the class label (digits, 0-9). Each column after the first was a single ith-row and jth column pixel, containing the color of the pixel on a 0-255 scale. A single row was structured as follows:

$$[class\ label \quad pixel_{0,0} \quad pixel_{0,1} \quad pixel_{0,2} \quad \dots \quad pixel_{28,27} \quad pixel_{28,28}]$$

4. Experiments and Results

a. Distance-based feature extraction

The set of features extracted was based around its distance to the bounding box's edges, which surrounds the image. After binarization was applied to the images, and affine transformations were used to rotate the images upright to attempt to eliminate unnecessary variability, the distance between the four corners of the screen and the handwritten digit, going along the diagonal, were measured, as well as the distance between the midpoint of each side and digit.

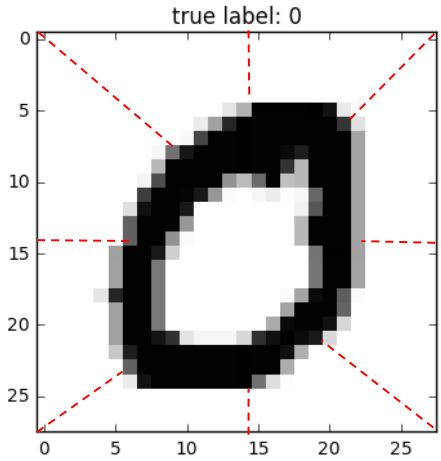


Fig. 4. An example of the 8 features extracted from an initial pattern.

Once the features were extracted from all the training patterns and testing patterns, they were written out to two new csv files, with each row organized as follows:

$$[up\ left \quad up\ mid \quad up\ right \quad right\ mid \quad bottom\ right \quad bottom\ mid \quad bottom\ left \quad left\ mid \quad class\ label]$$

b. Histograms for individual features

Eight histograms were generated to show off the separability between classes for each of the features, potentially highlighting which features may be better at distinguishing between classes. They can be seen on the next page. The middle right boundary feature seemed to have the best separability, along with the lower left boundary and middle lower boundary. However, some of the features had substantially higher interclass similarities than others, most notably with the middle upper boundary (outside of the digit, 4) and the upper left boundary. In these cases, many of the classes overlap. The digits, 8 and 9, seemed to be two of the hardest to distinguish between, as for every feature, they noticeably overlap with at least one other class. Some classes had high intraclass similarity, as well. The best example of this was the digit, 1, which, for every feature, tended to hover around a single value.

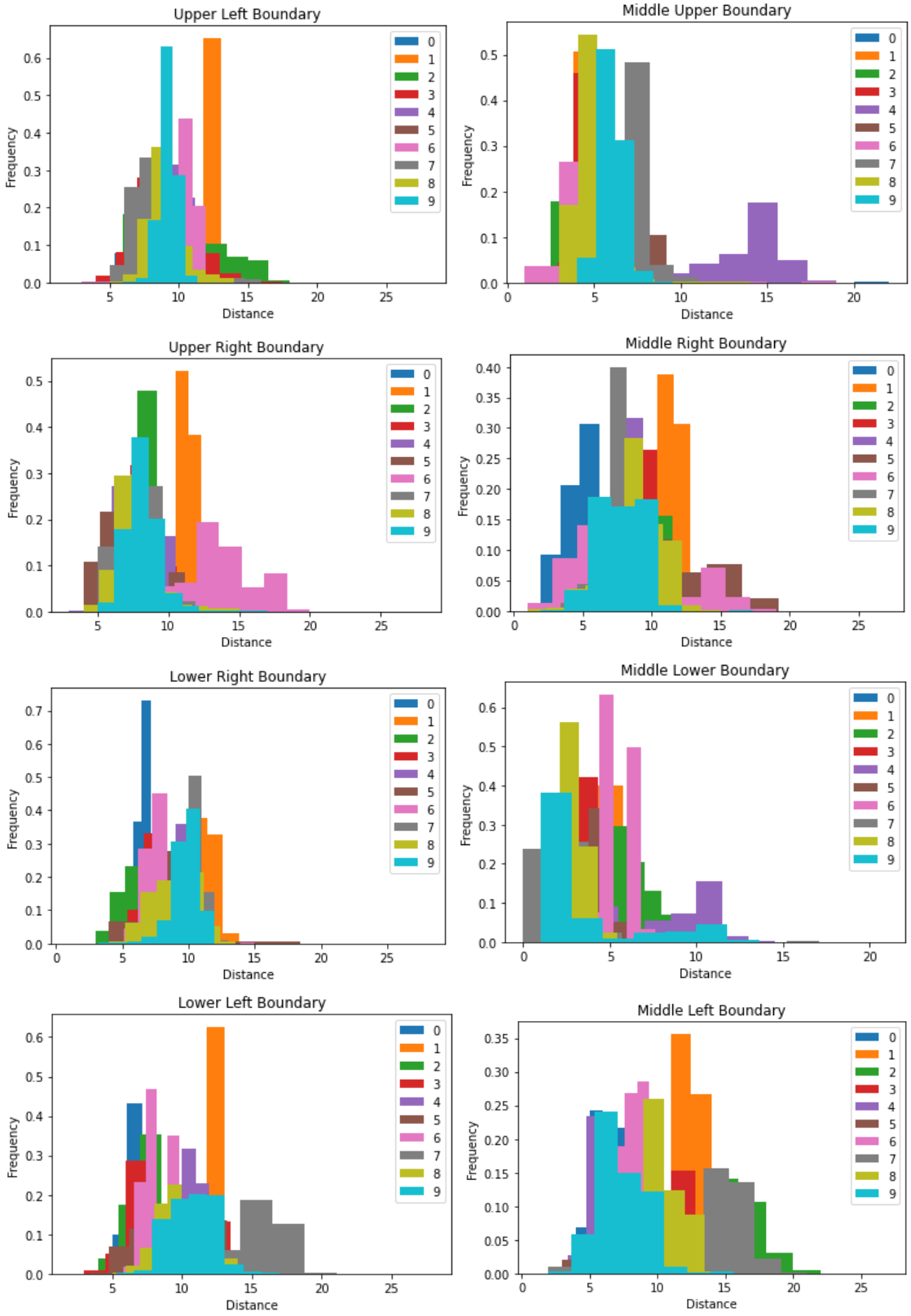


Fig. 5. – Fig. 12. Eight histograms, each pertaining to the 8 features

c. Finding the ideal K for K-NN

Next, both K-NN and Gaussian Parzen windows were tested out with subsets of the datasets to see which should be used. The ideal K for K-NN and the ideal window size for a Gaussian Parzen window were looked at, as well. Nine subsets of both the training data and testing data were generated. Each training subset contained 6,000 samples (600 patterns for each class). Each testing subset contained 1,000 samples (100 patterns for each class).

Starting with K-NN, the K values that were tested included 1, 5, 9, 13, 17, 21. Below, the error rates for each subset given one of the six K values is shown, along with the average error rate and the variance of the error rate for each K.

- $\boldsymbol{k = 1}$: $\mu_{error} = .204 \ \ and \ \sigma^2{}_{error} = .0006$
- $\boldsymbol{k = 5}$: $\mu_{error} = .181 \ \ and \ \sigma^2{}_{error} = .0008$
- $\boldsymbol{k = 9}$: $\mu_{error} = .182 \ \ and \ \sigma^2{}_{error} = .00091$
- $\boldsymbol{k = 13}$: $\mu_{error} = .184 \ \ and \ \sigma^2{}_{error} = .00099$
- $\boldsymbol{k = 17}$: $\mu_{error} = .189 \ \ and \ \sigma^2{}_{error} = .0012$
- $\boldsymbol{k = 21}$: $\mu_{error} = .191 \ \ and \ \sigma^2{}_{error} = .001$

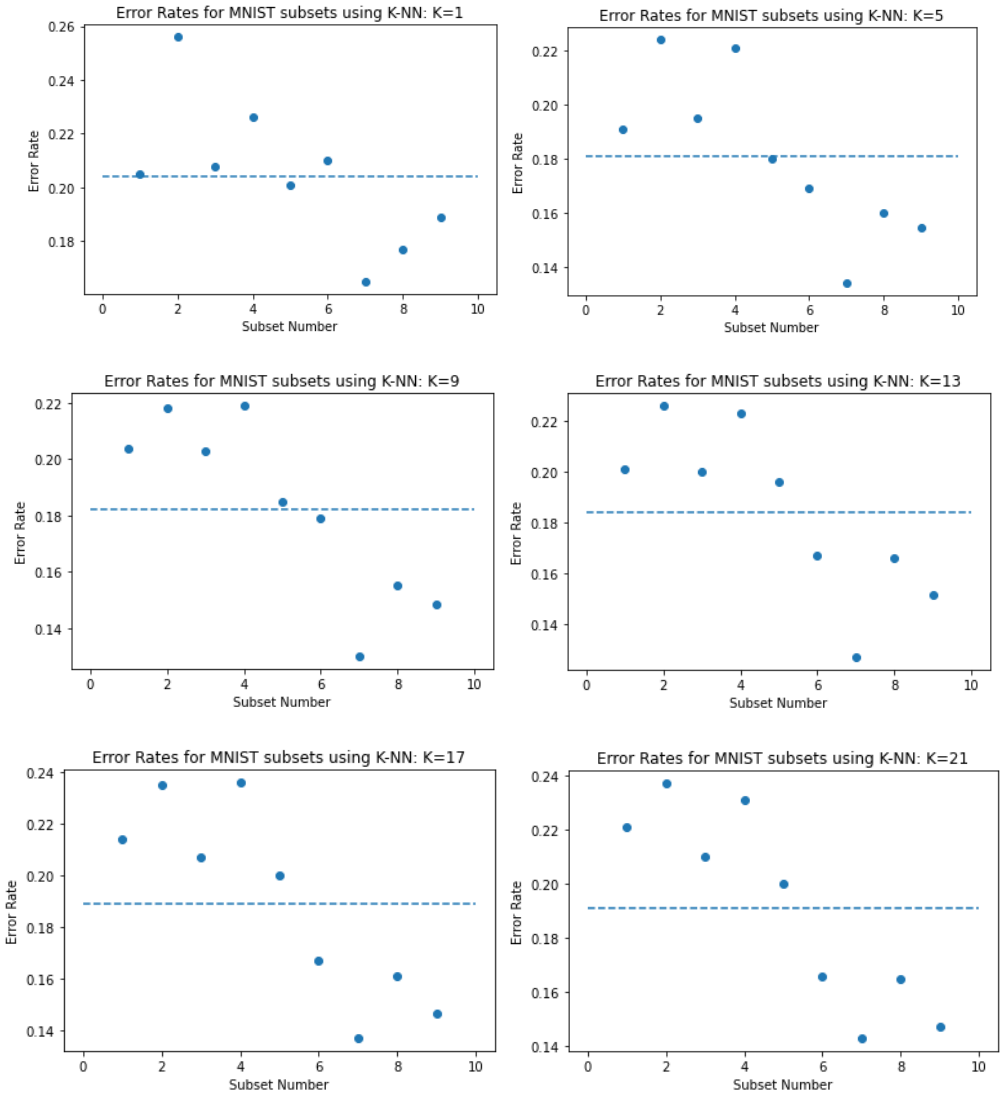


Fig. 13 – Fig. 18. The error rates for each subset, given different K sizes

d. Finding the ideal window size for Gaussian Parzen Windows

Next, different windows sizes for a Gaussian Parzen window were tested to determine what was the ideal window size. Window sizes of .1, .25, .5, 1, 1.5, 2 were tested to determine the ideal window size, using the same subsets as K-NN. Below, plots of the error rates for each subset given one of the six window sizes is shown, along with the average error rate and the variance of the error rate for each window size.

- $\boldsymbol{h = .1}$: $\mu_{error} = .1875$ and $\sigma^2{}_{error} = .00065$
- $\boldsymbol{h = .5}$: $\mu_{error} = .1781$ and $\sigma^2{}_{error} = .00067$
- $\boldsymbol{h = 1.5}$: $\mu_{error} = .2229$ and $\sigma^2{}_{error} = .00134$
- $\boldsymbol{h = .25}$: $\mu_{error} = .1797$ and $\sigma^2{}_{error} = .00069$
- $\boldsymbol{h = 1}$: $\mu_{error} = .1823$ and $\sigma^2{}_{error} = .00077$
- $\boldsymbol{h = 2}$: $\mu_{error} = .2653$ and $\sigma^2{}_{error} = .00179$

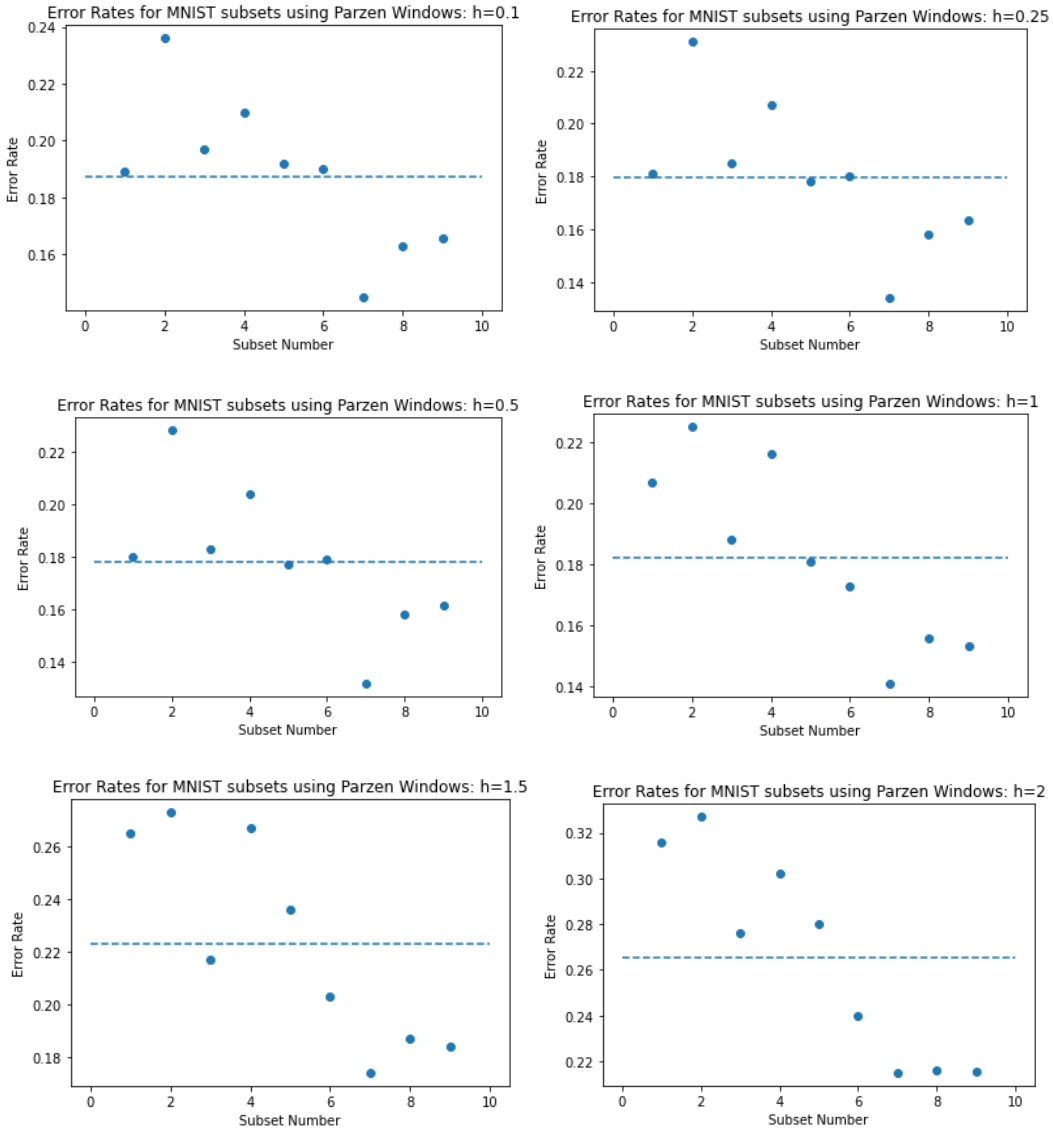


Fig. 19 – Fig. 24. The error rates for each subset, given different window sizes (h)

When using K-NN, there was little variability between different K values, outside of using 1-NN, which while yielding the highest average error rate between the subsets, its variance was the lowest by a small margin. However, as **K=5** yielded the best mean error rate, despite the difference being minimal, and its variance for the error rate was the second smallest behind K=1, it will be used when testing the K-NN classifier on the entire dataset.

When using Gaussian Parzen windows, the smaller window sizes (.1, .25, .5, 1) appeared to do better than the larger sizes (1.5, 2). The window size that will be used to test the overall effectiveness of the classifier will be **h=0.5**. This was the window size that yielded the lowest mean error rate and second lowest error variance, insignificantly larger than the lowest error variance, making it the best choice to test on the entire dataset.

e. MDA

MDA was used on the datasets to reduce this problem's dimensionality. This would make the problem easier to work with and visualize. However, as the datasets are large and can be computationally intense to work with, this was tested on a smaller subset of data. Using both a window size of .5 for the Gaussian Parzen Window and a K of 5 for the K-nearest neighbors (as these were the best results yielded in the testing above), the features of the training and testing sets were transformed from 8 features to 2 features, plotted, and then tested to see how its error rate compared with the unaltered data. Below, the transformed feature space can be seen for both the training dataset and the testing dataset, using a training subset size of 12,000 samples (1,200 for each class) and testing subset of 2,000 samples (200 for each class).

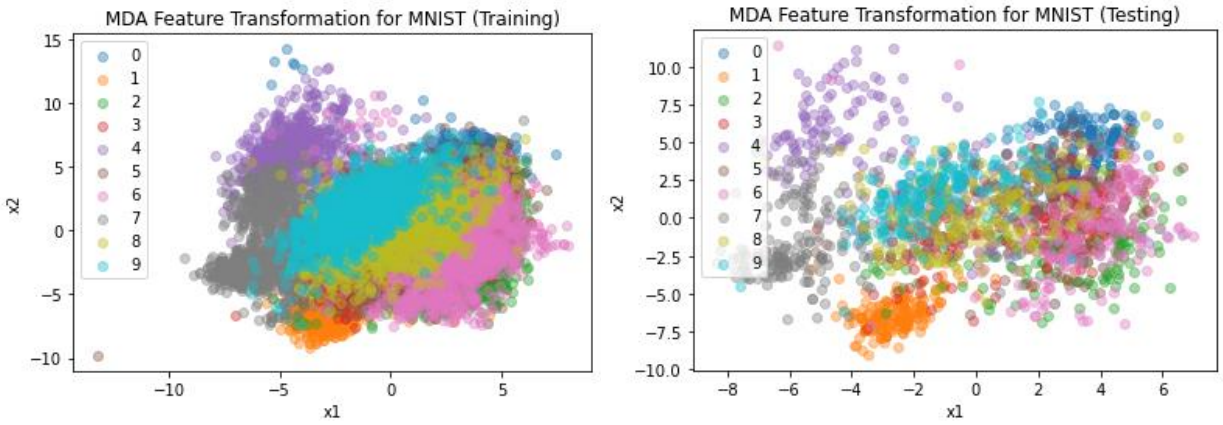


Fig. 25 and Fig. 26. A distribution of the training and testing feature-space after MDA was applied to each sample to reduce its dimensionality from 8 to 2, respectively.

While applying MDA does help visualize the distribution and reduces the complexity of dealing with a higher dimensional space, both K-NN and Parzen Windows yielded a higher error rate (>30%), performing substantially worse than its 8-dimensional counterpart. Thus, it appears that by projecting the number of features down to 2 dimensions, too much information is lost, and the classifier preformed significantly worse, discarding itself as a viable option for this classifier.

f. Overall

Once an ideal K value and window size were found, the entire dataset could then be tested with the classifier. There were two reasons as to why this was the case. For one, it is extremely computationally intensive, both in terms of memory and time complexities, to test using all 70,000 datapoints. For example, running K-NN for a single test point requires calculating the distances for all 60,000 training points, followed by then finding the K nearest neighbors. The other problem is that, while the single error rate it gives provides a good overall analysis of the classifier, it is not as detailed, being a single data point. The solution, which was used previously when comparing error rates between different K values and window sizes, was to split the training and testing samples into 9 smaller, equally sized subsets, where every subset maintained the ratio 6 training points per 1 testing point, maintaining the ratio of the original dataset. Not only was this a quicker way to test the points, but it also provided multiple error rates that could be compared with each other. A mean and variance was generated from them, along with figures that helped detail the effectiveness of the classifier. Based on that data, a single run through the entire datasets using K-NN and Gaussian Parzen windows was done using their respective ideal size, generating the results below.

First, a 5-NN classifier was used, and it yielded an error rate of **15.5%** along with its confusion matrix, which can be seen below.

```
[[ 962    0    2    2    3    2    5    0    3    1]
[   6 1112    6    2    1    1    6    0    1    0]
[  20    6  927   46    4    2    4   11   10    2]
[  26   14   57  824    1   36    3    7   39    3]
[  13    6    3    2  879    2    5    2    9   61]
[  28    3   10  101   10  686   18    2   29    5]
[  39    5    0    4    6   30  872    0    2    0]
[   2    5   16   17   13    4    0  942   10   19]
[ 107   11   19  174   55   77   11    6  508    6]
[  17    5    4   11  181    7    1   18   24  741]]
```

Fig. 27. The confusion matrix for using 5-NN on the entire training and testing datasets. The columns are the predicted values, and the rows are the actual values.

Table 1. The classification accuracy of each individual digit using 5-NN

0	1	2	3	4	5	6	7	8	9
98.1%	98%	89.8%	81.6%	89.5%	69.9%	91%	91.6%	52.2%	73.4%

Next, also using the entire dataset, a Gaussian Parzen window with a window size of 1 was used. Its error rate was **13.6%,** and its confusion matrix can be seen in figure 6.

```
[[ 949    0    3    3    2    3   10    0    7    3]
 [   2 1112    3    3    0    2    6    1    6    0]
 [  13    7  896   64    3    4    4   14   23    4]
 [  18   12   32  769    0   59    8   11   95    6]
 [  11    7    1    2  797    3    9    1   16  135]
 [  23    1    4   58    8  683   26    3   81    5]
 [  26    4    0    1    3   12  903    0    6    3]
 [   2    2   10   12    6    3    0  951    9   33]
 [  69    8   12   84   24   56    9    7  665   40]
 [  10    4    4    6   38    5    1    7   19  915]]
```

Fig. 28. The confusion matrix for using a Gaussian Parzen window with a window size of .5 on the entire training and testing datasets. The columns are the predicted values, and the rows are the actual values.

Table 2. The classification accuracy of each individual digit using Gaussian Parzen window with a window size = 1

0	1	2	3	4	5	6	7	8	9
96.8%	98%	86.8%	76.1%	81.2%	76.6%	94.3%	92.5%	68.3%	90.7%

For both confusion matrices, it appears that the digit, 0, and the digit, 1, had the highest classification accuracy. As the shapes of these two digits are very distinct, these results are not surprising. For digits, 2 though 4, K-NN seemed to perform slightly better than the Parzen window technique. For digit, 5, both classification models performed relatively poor. This most likely is due to its being misclassified as a 3 or 8, where the middle left distance might not pick up on the distinctions for the left side of each digit.

Digits, 6 and 7, both returned high classification accuracies, while 8 ended up being the worst in both models. There was too much interclass similarity between 8, 3, and 0. Based on the 8 extracted features, the shapes of these three digits can be very similar, most likely attributing to these poorer results. Finally, the digit, 9, fared noticeably better for the Gaussian Parzen window method.

f. Conclusion

Overall, the effectiveness of this classifier, while not as accurate as intended, was still successful. For K-NN with a neighbor (K) size of 5, its classification accuracy was about 85%. For the Gaussian Parzen Window with a window size of .5, its classification accuracy was about 87%. Compare this to the Convolutional Neural Network (CNN) classification method for the MNIST dataset, which according to [4], yielded a classification accuracy of over 99%, the results seem poor. However, the intent of this project was not to develop a better classifier than the CNN classification model; it was to see if developing a simpler, non-parametric classification model could yield results not too dissimilar from the current standard.

In this model, preprocessing was minimal. Before feature extraction, binarization was applied to all the pattern's pixels. Following this, affine transformations were applied to de-skew each of the images, reducing some of the variability in how the digits were written. The feature extraction process was also minimal. Distances from the midpoints and corners of a 28 x 28 bounding box to handwritten characters were calculated. Thus, each sample's 784 features were reduced to 8, and each of these 8 features were simple distance metrics.

However, there were still flaws with the feature extraction. For example, the digit, 8, seemed to commonly be confused with the digit, 0, or the digit, 3. This can most likely be blamed on the features extracted, as these 8 features can be very similar between the 3 digits, and, in retrospect, other features not related to distance could have been explored to help distinguish between these 3 classes (such as pixel density for certain quadrants of each image). The digits, 5 and 3, tended to be wrongly classified as each other, as well, most likely due to the same reasons above.

Despite this, this project was still successful. Each pattern, which started out complex, was simplified with a significant loss of information. Despite this, the resulting features and estimation-based classification model led to a small error rate of 13.6%, meaning that 86.4% of handwritten digits were correctly identified. While more advanced classification models, such as CNN or SVM (Support Vector Machines) exist, they involve complexity and overhead well above what this classifier required for its success. So, achieving this level accuracy shows that simpler classification models for the MNIST dataset can still achieve fairly accurate and effective results, and with fine-tuning of the feature extraction process for this model, it can achieve results even closer to that of CNNs or SVMs.

References

[1] *MNIST Dataset*, Year of Publication. Accessed on: April 15 2021. [Online]. Available: https://pjreddie.com/projects/mnist-in-csv/

[2] S. Raschka, "MNIST Dataset," 2014-2020, *GitHub*. [Online]. Available: http://rasbt.github.io/mlxtend/user_guide/data/mnist_data/, Accessed on: April 16 2021.

[3] (2016) Deskewing[Source code]. https://fsix.github.io/mnist/Deskewing.html

[4] Shivam S. Kadam, Amol C. Adamuthe* , and Ashwini B. Patil, “CNN Model for Image Classification on MNIST and Fashion-MNIST Dataset” Journal of Scientific Research Volume 64, Issue 2, 2020. [DOI: 10.37398/JSR.2020.640251]. Available: https://new.bhu.ac.in/Images/files/51.pdf. [Accessed April. 22, 2021].