Classification of Handwritten Digits Using KNN Classifier

By Raymond Ma

MNIST Dataset:

The dataset that I am using for this project comes from the MNIST database which contains 60,000 digits images, ranging from 0-9, that will be used to train our algorithms. The database also contains another 10,000 digits images that I will use as test data. Each image in this dataset is a digit centered in a 28 x 28 gray image, which means each image has 784 pixels of features.

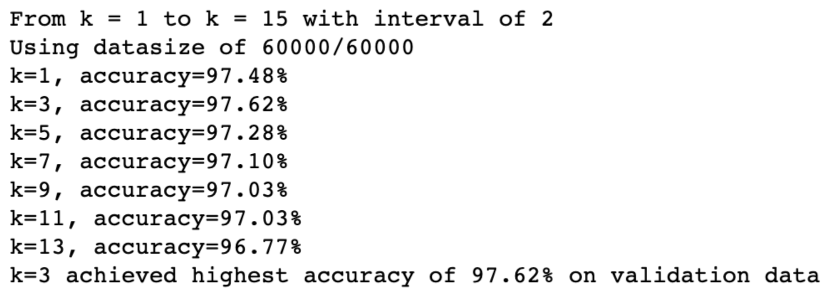
K-Nearest Neighbors Algorithm:

The algorithm that I used to recognize handwritten digits from the MNIST dataset is via the k-Nearest Neighbors Classifier (KNN). The KNN classifier is a simple image classification algorithm that relies on the distance between feature vectors to accurately classify unknown data points, digits in this project, by finding the most common class among the k-closest examples [1]. The most common class is determined by whichever class has the majority of the votes or most votes out of the k-closest neighbors. The KNN classifier generally uses the Euclidean distance to calculate distance between two data points [1]. To start our KNN classifier, I first took our MNIST dataset and changed the shape of the array that it was stored in so I could use it with our classifier. Next, I found the best k value to use for our KNN classifier.

I. Finding the Optimal K-Value

In order for the KNN classifier to work most accurately, I need to determine the best value for k that gives us the most accurate classification. Generally, if the k-value is too small, the algorithm will be more efficient due to having to account for less number of neighbors but might also lose accuracy as the model becomes too "complex", thus becoming more affected by noise and outliers [4]. If the value of k is too big, then I risk overfitting and our accuracy will decrease. Less emphasis will be put on individual points and our decision boundary will become simpler and more "smooth" [4]. Regardless of what the value for k is, the concept remains the same as the largest number of votes in k closest neighbor will determine the label used for that testing data point [1]. In our program, in order to determine the best k-value to use, I will test different values of k. First I randomly select 10% of our training data to serve as our validation dataset. Using these validation data points, I test them on the remaining training data at various values for k. I then compare the accuracy across all values of k that were tested and pick the most accurate k value for our actual test. During these tests on the validation data points, I only test values of k that are odd (k = 1,3,5 ..) because this will mean that there are never any “ties” between the votes.

After multiple tests to find the optimal k value, I discovered the best k value to test the MNIST data set [1]. As shown in our example run in Figure 1 below, the best k value is when k = 3. The general pattern is that as the value for k increased, the accuracy tends to decrease. As the number of training data points increases, the difference between the all the k values decrease as well.

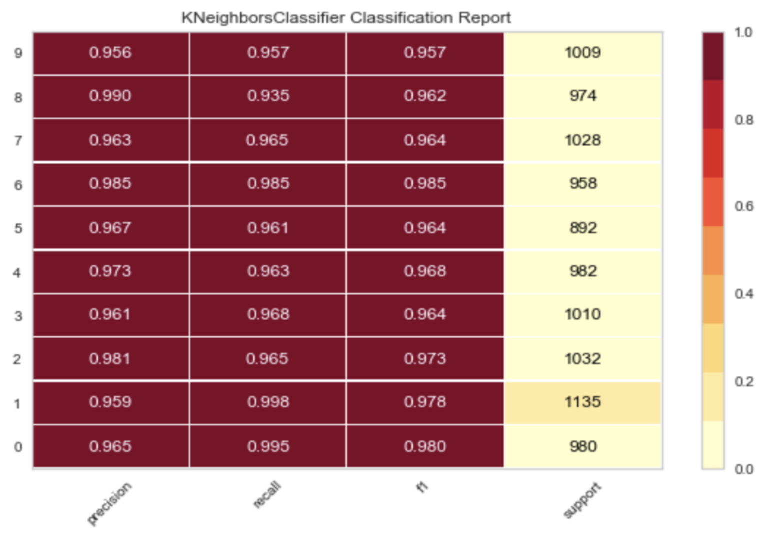


```
From k = 1 to k = 15 with interval of 2
Using datasize of 60000/60000
k=1, accuracy=97.48%
k=3, accuracy=97.62%
k=5, accuracy=97.28%
k=7, accuracy=97.10%
k=9, accuracy=97.03%
k=11, accuracy=97.03%
k=13, accuracy=96.77%
k=3 achieved highest accuracy of 97.62% on validation data
```

Figure 5

II. Results and Analysis

After finding the best k-value for our KNN classifier, I will use that k-value to begin testing our test data. I retrain the KNN classifier using our newly obtained k-value and then proceed to predict the labels for our test data [1]. After the predictions are calculated, I create classification reports in order to display our results. Figure 2 below shows a classification report for our testing data. For each digit, the report shows the precision, recall, and f1 score. Precision is the percentage of predictions made that were correct. Recall is the percentages of images that were predicted correctly for each digit. The f1 score is the weighted harmonic mean of precision and recall. The support column shows the actual number of images that correspond to each digit. As shown in Figure 2, our results were highly accurate as most columns were over 95% and reflect the classification report. After calculating the overall accuracy for predictions on the test data, I got an accuracy rating of 97%.


KNeighborsClassifier Classification Report

	precision	recall	f1	support
9	0.956	0.957	0.957	1009
8	0.990	0.935	0.962	974
7	0.963	0.965	0.964	1028
6	0.985	0.985	0.985	958
5	0.967	0.961	0.964	892
4	0.973	0.963	0.968	982
3	0.961	0.968	0.964	1010
2	0.981	0.965	0.973	1032
1	0.959	0.998	0.978	1135
0	0.965	0.995	0.980	980

1.0
0.8
0.6
0.4
0.2
0.0

Figure 6

In addition to the classification report, other graphs were also plotted to display the result including a class prediction error graph. Figure 3 below shows a class prediction error graph whose goal is to show how good the classifier is. For each digit, the graph shows the actual values of all images that were predicted as that specific digit. For example, if an image with the actual value of 4 were predicted as a 0, then it would show up on the column for 0. Looking at Figure 3, I can see that every digit has a high rate of correct predictions and a low rate of incorrect predictions. Every digit incorrectly predicted multiple other digits incorrectly at some point in the test. Overall, the graph shows that the classifier is accurate and produces low error rates.

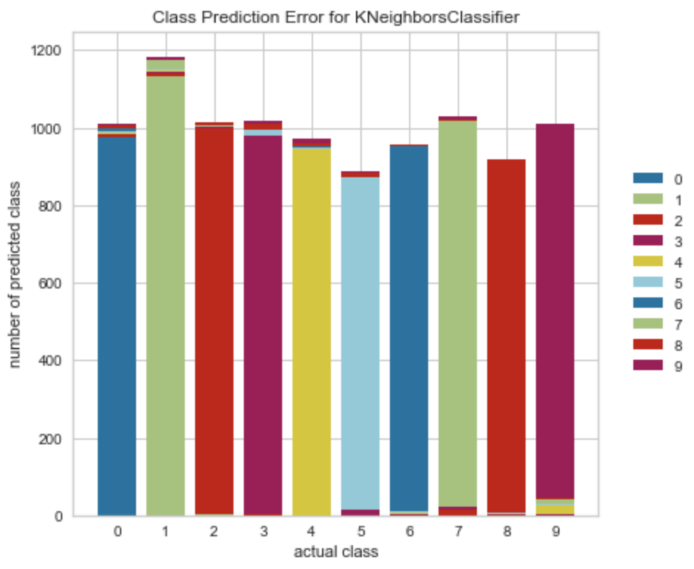


Figure 7

III. Final Thoughts on KNN Classifier

The KNN classifier algorithms are generally pretty simple and as such, the code for our KNN classifier system was also fairly simple. I first changed the shape of the MNIST dataset array from a 3 dimensional array to a 2 dimensional array. After that, I took 10% of our training data to use a validation data [1][2]. Then, using the validation data I just obtained, I tested them on the remaining training data using various values of k in order to determine which k gave the most accurate predictions [1][2]. After finding out the optimal k value, I began to predict the labels for our test data [1][2]. I displayed the results on various plots including classification reports and class prediction error graphs. I also was able to display the PCA pairs graphs between all the digits [5].

With an accuracy rating of 97%, the KNN classifier proves to be an effective algorithm to recognize handwritten digits. Looking at the PCAs plots for pairs of the MNIST dataset, there seems to be a clear distinction between any two digits which accounts for the high accuracy of the KNN classifier. A digit that stands out among the rest is the digit 1, when compared to every other digit, there is very little overlap. However, there are some digits where the opposite is true, such as the digits 4 and 9 which are shown to have heavy overlaps with one another. Overall, the PCA plots show that all digits are distinct enough from each other resulting in little overlaps, which helps the KNN classifier be more accurate. This is also the reason why the value for k during our testing was often very low, either k=1 or k=3. Since the digits are already so distinct from one another, a low number of closest neighbors was often good enough to make highly accurate predictions.

Bibliography:

[1] “k-Nearest Neighbor Classification.” *A Course and Community Designed to Take You from Computer Vision Beginner to Guru.*, gurus.pyimagesearch.com/lesson-sample-k-nearest-neighbor-classification/.

[2] “Handwritten Digits Classification(Using KNN).” *Kaggle*, www.kaggle.com/marwaf/handwritten-digits-classification-using-knn.

[3] Dalmia, Ayushi. “Handwritten Digit Recognition Using K Nearest Neighbor .” *Ayushi Dalmia*, researchweb.iiit.ac.in/~ayushi.dalmia/reports/Hand Digit Recognition.pdf.

[4] “8.3. Learning to Recognize Handwritten Digits with a K-Nearest Neighbors Classifier.” *IPython Cookbook, Second Edition*, ipython-books.github.io/83-learning-to-recognize-handwritten-digits-with-a-k-nearest-neighbors-classifier/.

[5] Mueller, Andreas. “Plotting PCAs of Pairs of MNIST Digit Classes.” *Github*, 15 Dec. 2012, gist.github.com/amueller/4299381.