9/20/2020

Classification of Digits

[Supervised Learning]

Zeyad Mohamed Bahgat

Table of Contents

Table of Figures

1 ABSTRACT

In machine learning, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs to, on the basis of a training set of data containing observations (or instances) whose category membership is known (known labels). Classifier performance depends greatly on the characteristics of the data to be classified. There is no single classifier that works best on all given problems.

A classifier can be assessed through his ability to **generalize**. Generalization refers to the classifier's ability to give accurate predictions for new (previously unseen) data. The trained model directly impacts the generalization of a classifier (classification algorithm). If the model is too complex, classifier will do well during training but bad during testing "overfit". While, if the model too simple or flexible, the classifier will not even do well during testing "underfit".

In this report, I have implemented one of the famous classification algorithms **KNN** (which stands for K-Nearest Neighbor) in order to classify images that belong to 10 digits (numbers from 0 to 9). During the implementation of this classifier, I tried to avoid the problems that might face the KNN classifier such as noises, also determined the optimum value of K to avoid overfitting and underfitting. The results of classification will be shown in the last section of the report.

2 INTRODUCTION

K- Nearest Neighbor is one of the simplest machine learning algorithms based on supervised learning techniques that can be used for both regression and classification. My main focus here is using the KNN in classification, where it tries to classify an unlabeled data based on the similarity of this data to the labelled data **stored** previously in the algorithm. Due to storing the labelled data without learning and using them at the time of classification, the KNN algorithm is also called a **lazy learner algorithm**.

The following points summarize the most important advantages and disadvantages of the KNN algorithm that can justify whether to use or not use the KNN algorithm for the classification of the 10 digits:

- Advantages)
    - Very simple and easy to implement.
    - KNN is a non-parametric algorithm, so it has no assumptions.
    - The flexibility to choose distance from a variety of distances (Euclidean, Hamming, Manhattan, etc.).
    - Does not learn anything during training (it is an instance-based learner), so it has no training period.
    - New training data can be added without impacting the accuracy of the algorithm
- Disadvantages)
    - KNN is computationally expensive as it searches the nearest neighbors for the new point at the prediction stage
    - High memory requirement as KNN has to store all the data points
    - Sensitive to noisy data, missing values, and outliers
    - K-NN needs homogeneous features (feature standardization and normalization)

The given training data set for classifying the 10 digits is not too large (about 2400 images) so I will not face any computation or memory problems and dataset is labelled. However, most of the images in the data set contain salt and pepper noise; but a simple median filter is more than enough to solve this problem as will be discussed in the following sections. So, during this task, implementing the KNN algorithm will be appropriate.

In the following sections I will explain in detail the process of classifying the test dataset, starting from preprocessing the training dataset till obtaining the results.

3 CLASSIFICATION OF THE DIGITS

To classify a given set of un-labelled data, I had to pre-process the given training dataset following many steps that will be discussed in the pre-processing section. After pre-processing and storing the training dataset, I implemented the KNN algorithm taking into consideration the impact of the choice of ‘K’ on causing overfit or underfit model. Finally, the results of classification will be shown.

3.1 DATA PRE-PROCESSING

Before starting KNN implementation, I had to prepare the given training dataset to be used in creating the model. The first step was the dataset entry, followed by filtering the data, and finally feature extraction.

3.1.1 Data entry

I have created a folder named “training” that contains 10 subfolders. Each subfolder contains the training images for a digit. This helped to importing all the images in order along with their labels. Also, we make sure that all imported images are of the same size “normalization”.

3.1.2 Filtering data

As previously said, the KNN is sensitive to noisy data. Fortunately, the noise was a salt and pepper noise that could be filtered easily with a median filter as shown in figure 3.1.2.1

Figure 3.1.2.1 the difference between filtered and unfiltered image

3.1.3 Feature extraction

Feature extraction is the process of transforming the input data into set of features. Features are distinctive properties of input patterns that help in differentiating between the categories of input patterns. The main objective of feature extraction is to obtain the most relevant information from the input data (28x28 pixels in an image as an example) and represent those in a lower dimensionality space(1x81 as an example), so the input data are transformed into a reduced representation set of features called feature vector.

There are many techniques used in feature extraction for classification, but the sole goal is the same which is creating a feature vector that can be used to distinguish between classes. In this task I used a ready-made function that extracts the features using the HOG “Histogram Of Oriented gradients” feature descriptor. A general overview about how HOG is implanted is shown in figure 3.1.3.1.

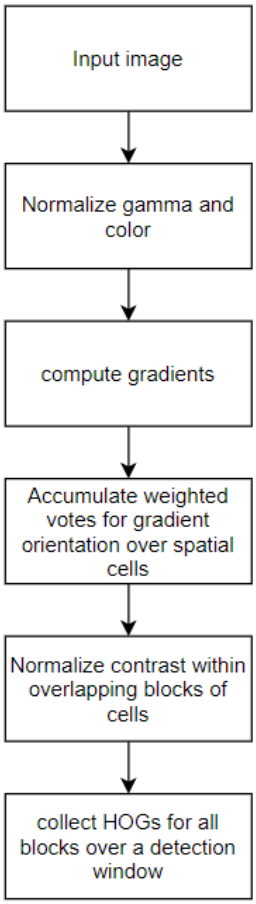

Figure 3.1.3.1 HOG general scheme

3.2 KNN IMPLEMENTATION

After the pre-processing stage, I now have a set of feature vectors and labels that I can use in the KNN algorithm. I will start with explaining how the KNN works to classify classes and the problems that might occur. Also, how to choose the optimum value for K.

3.2.1 KNN Algorithm

I will illustrate how the KNN algorithm works on a binary class instead of a multi-class classification example for simplicity. Suppose we have two classes of stars (each class consists of 12 stars) as shown in figure 3.2.1.1 and they are in a 2-d plane, so each star has a coordinate. Let us assume that each feature vector contains the coordinates of one training sample (x and y coordinates), and all these feature vectors are saved along with their labels (stating which feature vector belongs to which class).

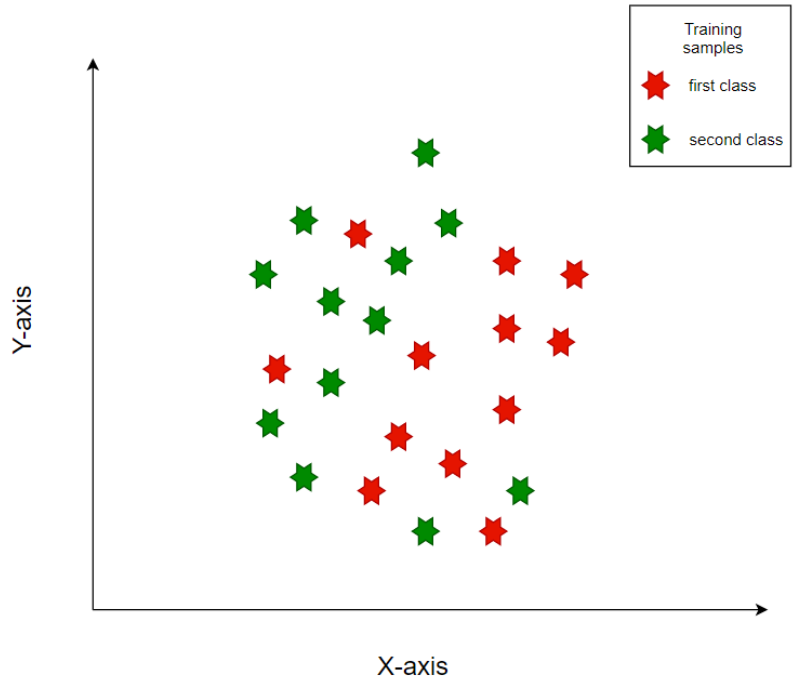

Figure 3.2.1.1 training samples

When a new star (test sample) is introduced as shown in figure 3.2.1.4, we want to know to which of the classes this star belongs to. To know this, we will follow the following steps:

1. Get the feature vector of the new star, which is in our case it's x and y coordinates.
2. Measure the distance between the test sample and all of the training samples: there are multiple ways to measure the distance, namely:
    - Euclidean Distance (which is the most popular)
    - Hamming Distance
    - Manhattan Distance
    - Minkowski Distance

    I used Manhattan distance (equation 2) in the code as it is slightly faster than the Euclidean distance (equation 1) in computation. Where F is the feature vector containing X and Y coordinates, Z is the elements inside each feature vector (in our case Z=2 "X and Y")

$$distance = \sqrt{\sum_{i=1}^{z}(F_{test}[i] - F_{train}[i])^2} \quad (1)$$

$$distance = \sum_{i=1}^{z}|F_{test}[i] - F_{train}[i]| \quad (2)$$

So, now we got distances between the test sample and all training samples as shown in figure 3.2.1.2.

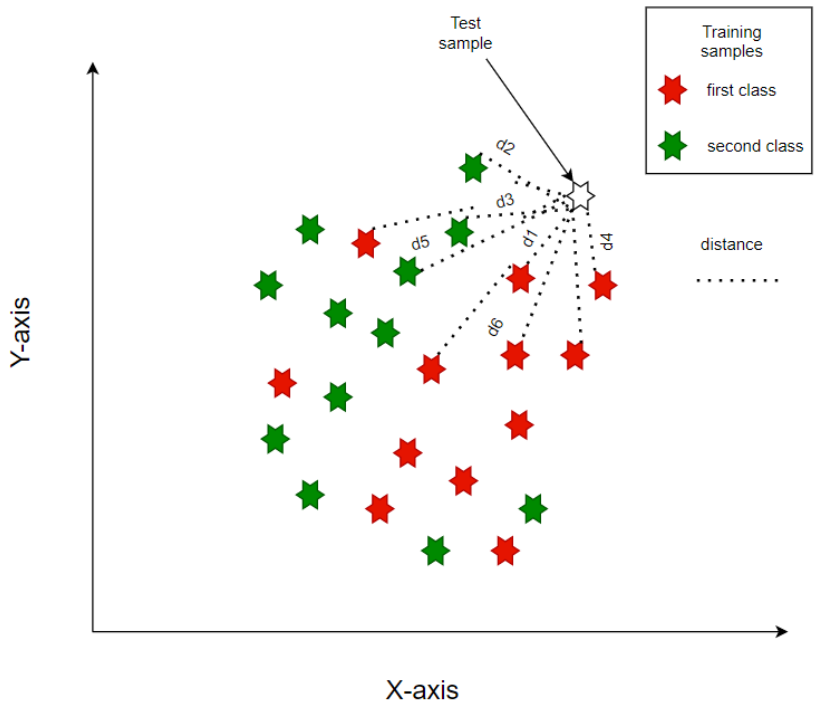

Figure 3.2.1.2 calculating distances

3. After knowing all of the distances between the test sample and the training samples, we will sort the distances from the minimum to the maximum distance. we also have the label (class of the star for which the distance between the test sample and the training sample is calculated) as shown in figure 3.2.1.3.

Classification	
Distances	**Label (0 is red, 1 is green)**
d1	0
d2	1
d3	1
d4	0
d5	0
. . . . .	. . . . .
dN	0

Figure 3.2.1.3 labels corresponding to the distances sorted

4. Now we will assume multiple scenarios to see the importance of choosing an appropriate k and then classify out test sample.

    - First scenario is choosing K=1, then the test sample would belong to the class with label of d1 which is “red”.

- Second scenario is choosing K=2, so we got two options, either it belongs to the class with label of d1 or d2. Because of this situation we try to avoid choosing K with an even number in binary classification as there might be no clear decision.

- Third scenario is choosing k =5, so we got 2 distances belonging to green star and 3 distances belonging to red star. So, the test sample would belong to the class with the most repeated label, in this case it belongs to the red star.

Hence, increasing the value of 'K' would give more accurate classification, but up to a limit. Assume that we had 36 training samples (12 green training samples 24 red training samples) and we choose k=30, we can notice that the classification result would be that the test sample belongs to red. That's because we would have 30 distances to see what is the most repeated label, which obviously will be the red, as if the first 12 labels belong to green, the other 18 labels would certainly belong to red. Resulting in classifying the test sample as red even if it was green, this is called underfitting.

In general, too small values of K would result in overfitting and too large values would result in underfitting

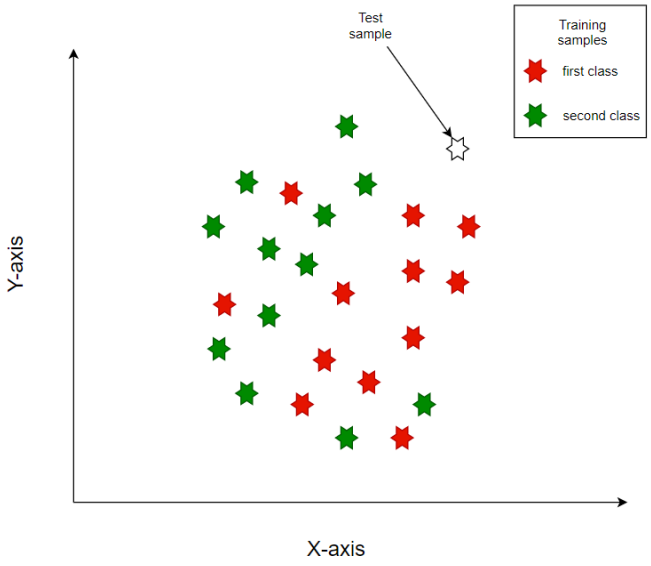

Figure 3.2.1.4 introducing test sample

I applied the same methodology in the multiclass classification of the 10 digits (will be explained in the following section), but as I previously said, the Manhattan distance was used instead of the Euclidean distance due to the faster computation.

3.2.2 Training

From the last section we should have understood the KNN algorithm and the importance of choosing an appropriate K value. SO, I will discuss how did I implement the multiclass KNN and determine optimum K value. To determine the optimum K value, I used the leave-one-out cross validation (LOOCV). figure 3.2.2.1 shows the process of LOOCV.

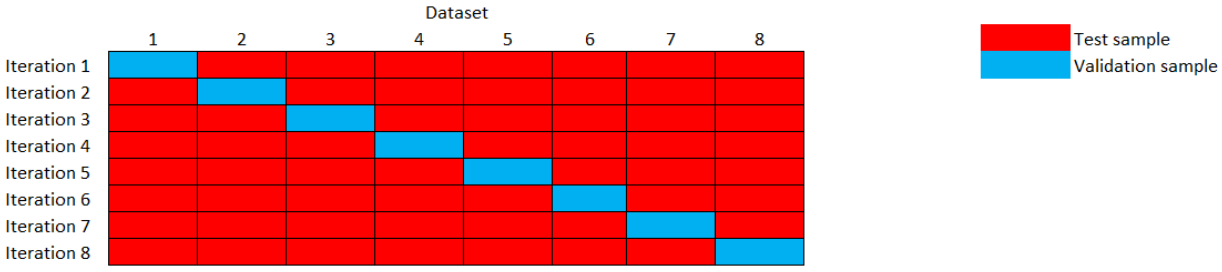

Figure 3.2.2.1 Iterations of LOOCV

Assume we have 8 samples in the dataset as shown in figure 3.2.2.1, the following steps are taken to determine the optimum K value using the leave-one-out cross validation:

1. I will compute the distances between each sample and the other samples, resulting in 64 distances (8 x 8) with 8 distances for each iteration. Those distances are saved.
2. For each iteration, I changed the K from 1 to 100 and recorded the accuracy in classification for each K (compare the classification result with the label. If the classification was correct then place ‘1’, while if it is wrong place ‘0’). Repeat this step for all the iterations.
3. Add the accuracy in classification for all the iterations recorded in each K and divide it by the number of samples (8). Repeat this step for all values of K.
4. The classification error for each K = 1- classification accuracy for each K.
5. The optimum K is the K with the least classification error.

The following steps simply will show how I implemented the KNN in training:

1. For each iteration, I sorted the saved distance of that iteration (the distance that I mentioned in step 1 above).
2. After sorting it, I repeated the following for K=1 till K=100:
    - Consider the minimum K distances and count the most repeated “frequent” label (the label that accompanies the distances, so we will also have K labels)
    - The most frequent label is the classification result

3.3 RESULTS

In this section I will show the training and testing results, as well as how did I use the KNN classifier in testing.

3.3.1 Training Results

The following figures will show the classification error versus the choice of K and the optimum K value when using and without using a median filter to remove salt and pepper noise. figure 3.3.1.1 and figure 3.3.1.2 show that the optimum k = 13, while figure 3.3.1.3 and figure 3.3.1.4 show that the optimum K = 4.

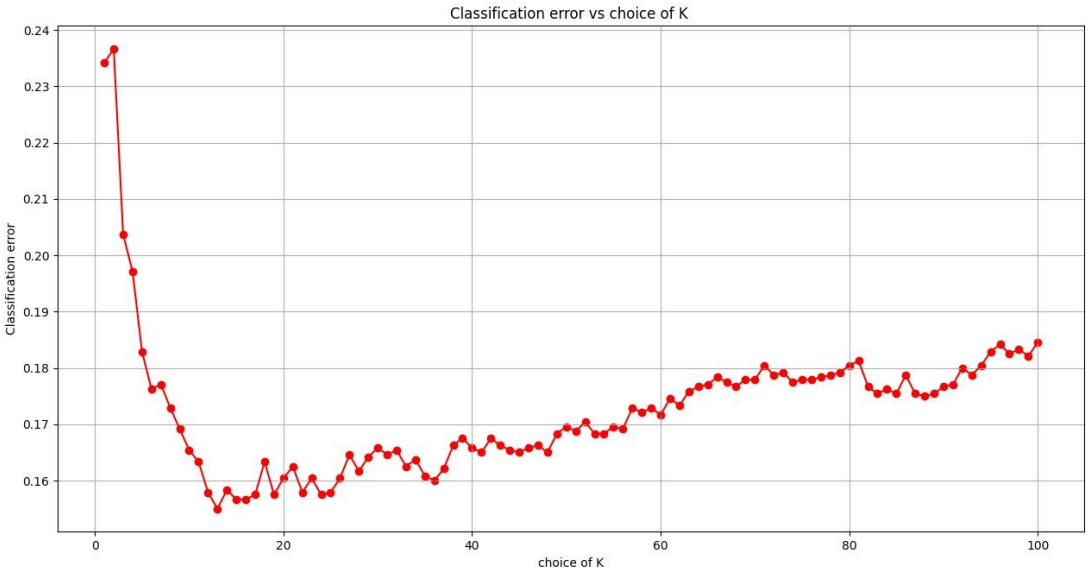

Figure 3.3.1.1 classification error versus choice of K without using a filter

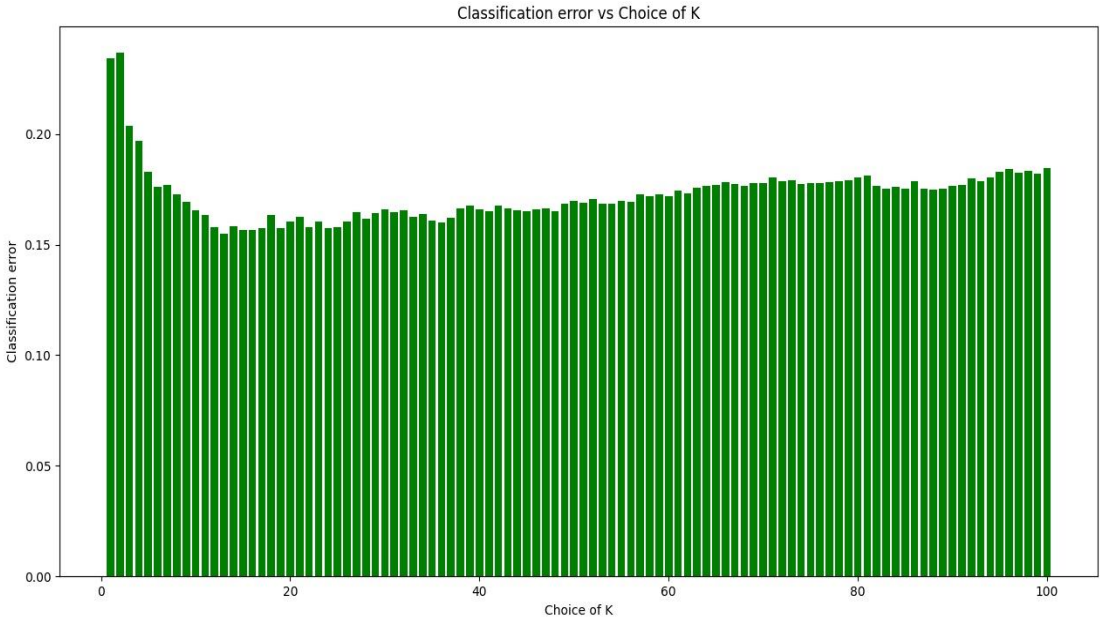

Figure 3.3.1.2 optimum choice of K without using filter

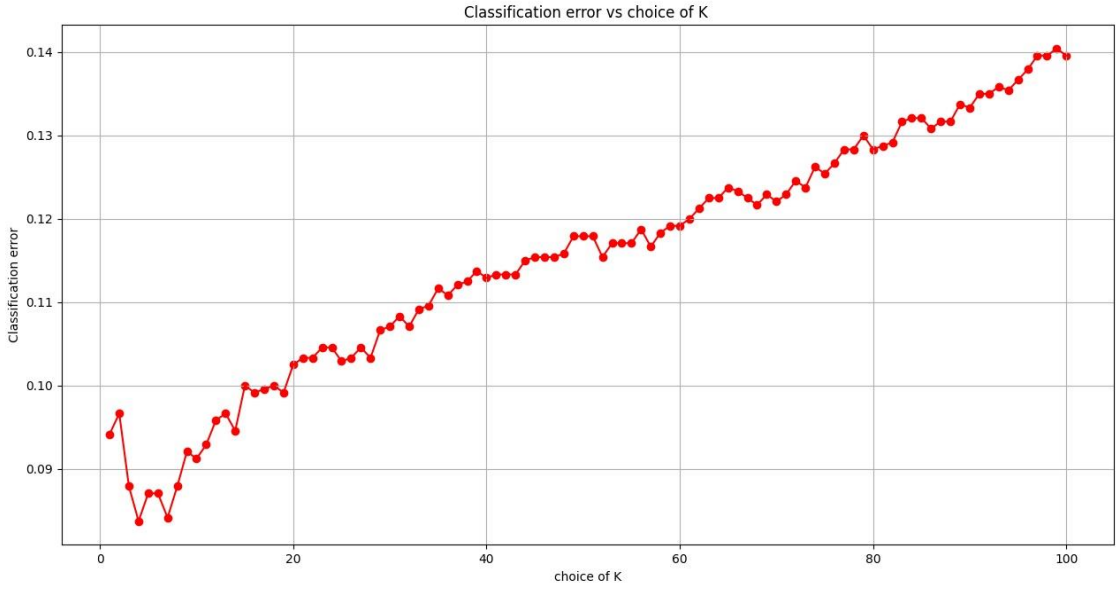

Figure 3.3.1.3 classification error versus choice of K when using a filter

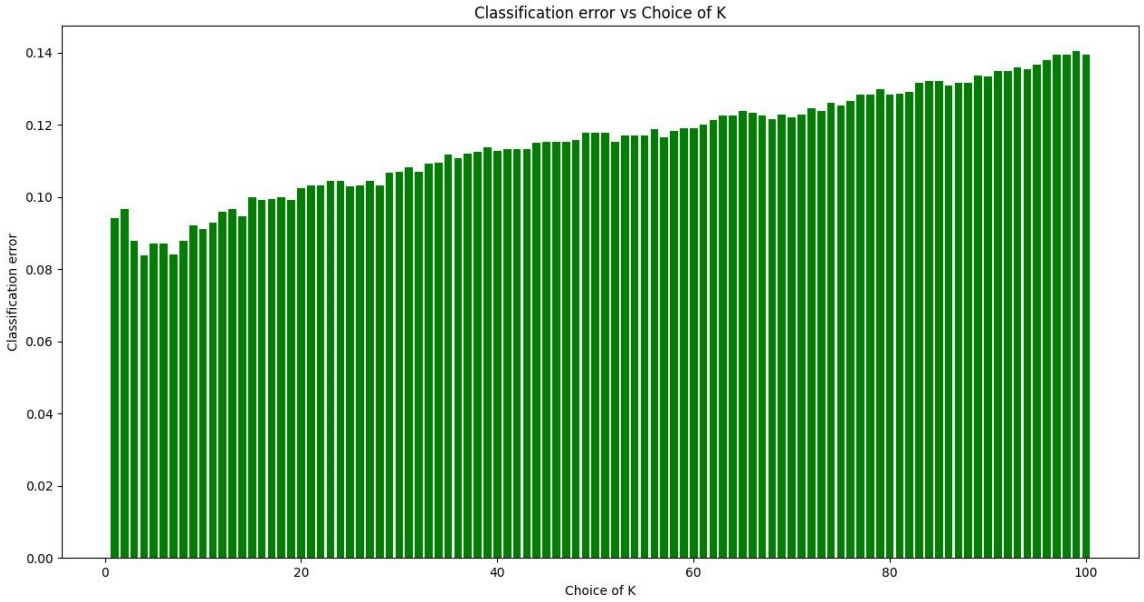

Figure 3.3.1.4 optimum choice of K when using filter

3.3.2 Testing Results

During testing I calculated the distances between the feature of a given test image and all of the saved training features, then sent the distances array to the KNN classifier to return the classification result. The following figures will show the confusion matrix of tested images (each digit) when using and without using the median filter.

figure 3.3.2.1 shows that the classification accuracy without using a filter and at the optimum K equals 0.85, while figure 3.3.2.2 shows that the classification accuracy when using a filter and at the optimum K equals 0.935

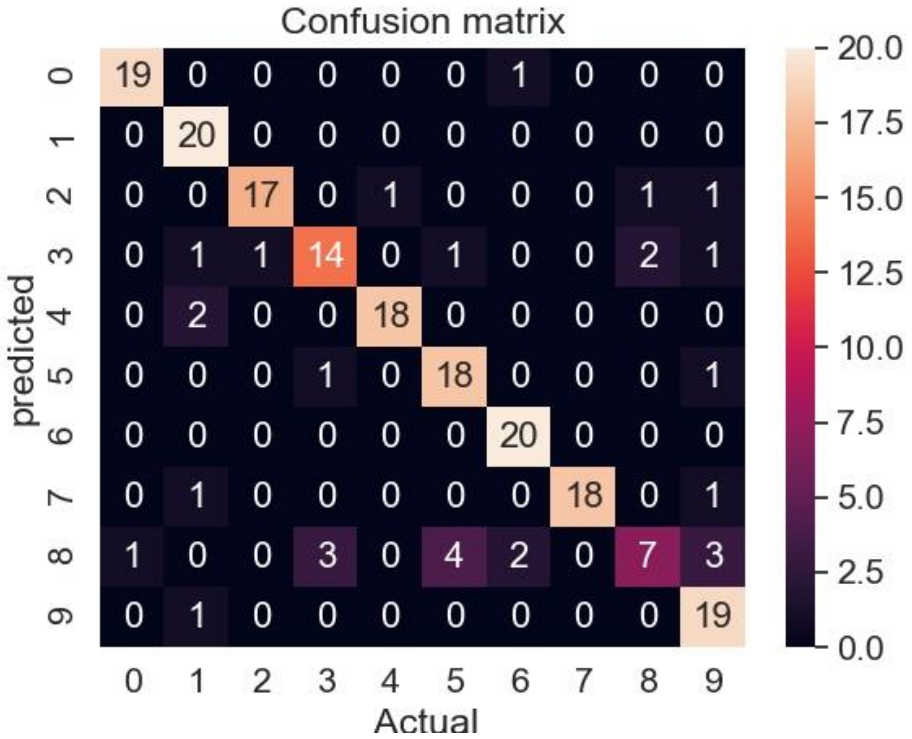

Figure 3.3.2.1 confusion matrix without using filter at k=13

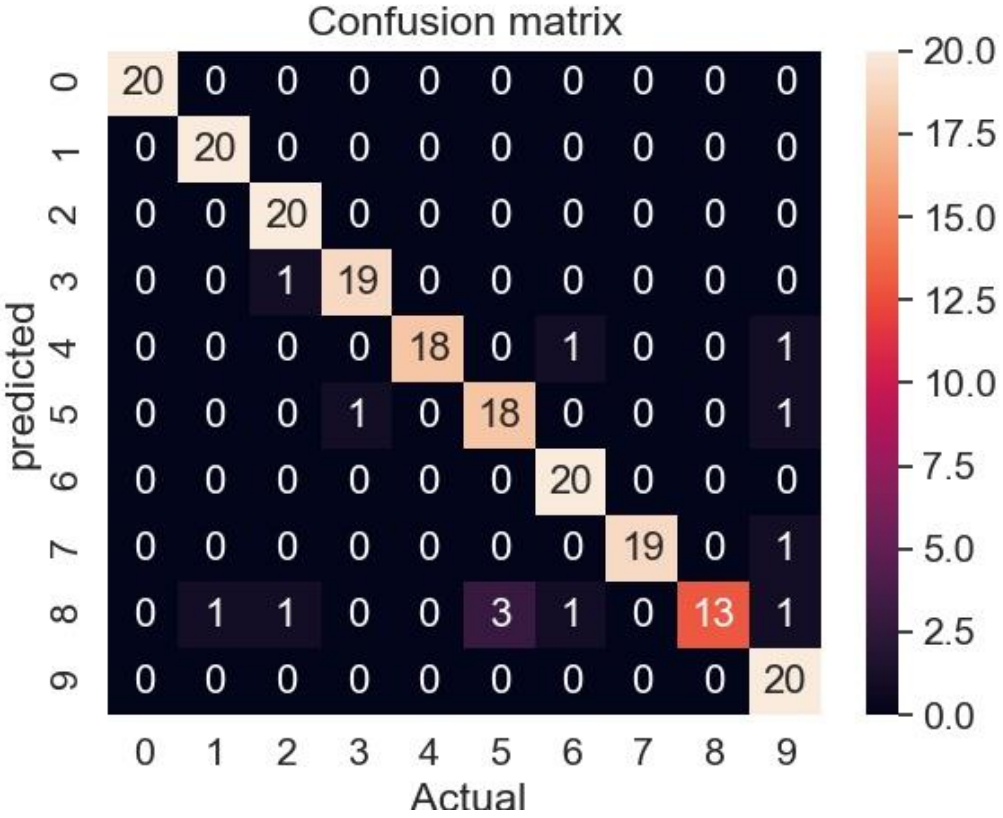

Figure 3.3.2.2 confusion matrix when using filter at k=4

4 CONCLUSION

The previous results show a general trend of decreasing the classification error when increasing the choice of K up to certain limit “optimum k”, then the classification error starts to increase again. This is due to the fact of overfitting at low values of K and underfitting at high values of K as previously explained.

Also, we can notice that when using a filter to remove the salt and pepper noise, the classification error for each choice of K decreased significantly (proving that KNN is sensitive to noise) and the accuracy of classification during testing increased from 0.85 to 0.935 as shown in the previous confusion matrices.

In general, I support using the KNN algorithm because it was not computationally demanding or taking much space or predicting the class slowly as the **dataset was relatively small.** The only problem, as I stated previously, was the presence of salt and pepper noise that has been removed by a median filter. Also, the overall accuracy ‘0.935’ for this size of dataset which is not bad.