

January Submission Assignment

Huazhi Lin
2200376

(of Affiliation)

Software Engineering and Artificial
Intelligence program
2200376@student.uwtsd.ac.uk

Abstract—Machine learning classification has traditionally been a research hotspot for handwritten digit recognition. In this study, two handwritten digit classification models based on the KNN and decision tree algorithms are developed. This article's construction uses the widely used machine learning library scikit-learn, and its validation is done using the MNIST dataset. In the preprocessing stage, the number set is divided into a training set and a test set in the ratio of 7:3. The best performance is then attained by experimenting with various parameters, and the two best performance classifiers are then compared. The performance of the model is then assessed using metrics like the confusion matrix, recall, precision, F1 score, and classification accuracy.

Keywords—machine learning, K Nearest Neighbors, Decision Tree

I. INTRODUCTION

For many artificial intelligence applications, such as automatic recognition of postal codes, automatic recognition of phone numbers on courier bills, automatic recognition of license plate numbers, and other features, handwritten digit recognition is a fundamental technique. As a result, several academics working in the field of artificial intelligence have conducted research on this in recent years. The template matching approach, the use of transformation features, the projection histogram method, and the geometric matrix feature method are the most used techniques for pattern statistical recognition [1]. Although these conventional statistical techniques can partially realize handwritten digit recognition, their recognition accuracy is poor and their ability to adapt to changes in the shape and direction of the digits is weak. When the numbers tilt, the background is intricate and the thickness varies significantly. The impact of recognition will drastically diminish.

Convolutional neural networks are employed for the recognition of handwritten digits as deep learning technology has advanced. Convolutional Neural Network (CNN) is very good at recognizing the structure of handwritten characters or words, which enables automatic feature extraction. [2] The accuracy rate of MNIST utilizing CNN is up to 99.8%. Bengio [3] used a deep network model to analyze a vast collection of handwritten digits in 62 various settings. It is demonstrated that the deep multi-layer network model can effectively extract the key properties of handwritten digital images through statistical theory and numerous simulation results.

Machine learning has gained a lot of attention as the foundational technology of artificial intelligence, and its performance in pattern identification has surpassed that of conventional statistical pattern recognition techniques. This study investigates and develops handwritten digit recognition, verifies it on the common handwritten digit picture dataset MNIST[3], and then compares the two using the traditional machine learning algorithms KNN and decision tree.

II. KNN ALGORITHM INTRODUCTION

A. KNN algorithm principle

The K closest neighbor algorithm [4] is another name for the KNN algorithm. Cover and Hart first suggested this technique in 1967. It is capable of both regression and classification. Consider the classification algorithm used by KNN as an example. The working idea of the algorithm may be interpreted as follows: given a training data set, discover the K instances in the training data set that are most similar to the instance in the new input data set. The input instance is allocated to a class if it falls under that class.

Figure 1 shows that the sample examples are split into two groups: the group symbolized by the triangle and the group symbolized by the six-pointed star. Below are the steps for utilizing the KNN algorithm to categorize the new sample circle. Assume a value for K first. If K is equal to 3, choose the two closest triangles and a six-pointed star. The circle should then be categorized in the same category as the two triangles. If K is equal to 5, choose the three closest six-pointed stars and the two closest triangles. The circle's samples should then be categorized in the same category as the six-pointed star

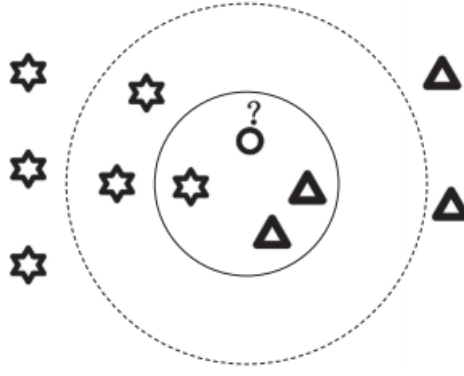


Fig. 1. Schematic diagram of the KNN algorithm

Finding the K samples that are most similar to the sample to be predicted is one of the two challenging difficulties with the KNN method. If the K value is too small for the first question, the prediction value won't be accurate enough, and if it's too large, the prediction procedure will be too difficult. Consequently, using the cross-validation method is the traditional way to choose the K value. The samples in the verification set are used to continuously change the value of K to discover the best value. A portion of the data in the training set is utilized as the training set, and a second portion is used as the verification set. Another approach is to iterate through the detection results produced by K and choose the best K value based on the best result value.

Finding the K samples that are most similar to the sample that needs to be forecasted is the second issue. How to determine the distance between two samples is the solution to this issue. At the moment, the most used methods for estimating distance include Manhattan Distance and Euclidean Distance. By the circumstances, you can also create your distance computation algorithm. Equations 1 and 2 lists the formulas for calculating the Manhattan distance and the Euclidean distance, respectively, using two nodes in a two-dimensional space (x_1, y_1) and (x_2, y_2) as an example:

$$EuclideanDistance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (1)$$

$$ManhattanDistance = |x_1 - x_2| + |y_1 - y_2| \quad (2)$$

We can determine the distance in this manner. The simplest and most basic KNN method involves measuring the separation between the predicted point and all other points, saving and sorting the results, and choosing the first K values to determine which categories are more prevalent.

B. Advantages and disadvantages of the KNN algorithm

- (1) It's commonly employed. KNN is a member of the larger category of techniques known as instance-based learning, which uses particular training examples to make predictions without needing to keep up a model based on the data.
- (2) It is more accommodating in cases where the sample categorization boundary is atypical. After extensive testing, it is discovered that the KNN algorithm is appropriate when the boundary of the sample categorization is atypical. The KNN algorithm is more effective than other techniques for dividing sample sets where the class domains overlap or overlap more because it primarily focuses on the limited surrounding samples rather than the way of differentiating the class domain to determine the category to which it belongs.
- (3) The calculations use low computation efficiency and a sizable amount of resources. For each test point, KNN must determine the distance to each training data point, and these distance points take into account all features. The calculation efficiency will be quite low if both the size and volume of the data are huge.
- (4) It is more sensitive to noise data since the noise immunity is inadequate (outliers). KNN classifiers (when k is small) are particularly susceptible to noise because of the local classification decisions it makes based on local input.
- (5) The repeatability of the model is poor, and it is unstable. Because they depend on the combination of training instances, KNN can generate decision boundaries of any shape. These decision boundaries have a great degree of unpredictability. This fluctuation can be decreased by raising the number K of nearest neighbors.

III. INTRODUCTION TO DECISION TREE ALGORITHM

A. Principle of decision tree algorithm

JBoss Quinlan created the decision tree algorithm [5] and first put it forth in 1960. This algorithm chooses the attribute with the most information gain as a split node to produce a decision tree and utilizes it as the foundation for attribute division. Starting with the root node, it tests the associated feature characteristics in the object to be classified, chooses the output branch based on its value, and continues until it reaches the leaf node, where it uses the category stored there as the decision result.

The ID3, C4.5, and CART algorithms are the three most widely used methods for creating decision trees. The CART method is used in this article to build the decision tree. The Gini index serves as the benchmark for choosing the best partition feature in the CART algorithm. The uncertainty of the set D is represented by the Gini index $Gini(D)$, and the uncertainty of the set D divided by $A=a$ is represented by the Gini index $Gini(D, A=a)$. The smaller the Gini index, the less uncertain the sample is. Given that there are K classes in the classification issue and that p_k is the probability that the sample point belongs to class K, the Gini index of the probability distribution is defined as:

$$Gini(p) = \sum_{k=1}^K p_k(1 - p_k) = 1 - \sum_{k=1}^K p_k^2 \quad (3)$$

If the feature A takes one of several possible values, such as a, and the sample set D is partitioned into two sections, D1 and D2, then:

$$D_1 = \{(x, y) \in D | A(x) = a\}, \quad D_2 = D - D_1 \quad (4)$$

The Gini index of set D is thus defined as follows when feature A is satisfied:

$$Gini(D, A = a) = \frac{|D_1|}{D} Gini(D_1) + \frac{|D_2|}{D} Gini(D_2) \quad (5)$$

The CART algorithm's steps are:

Training data set D as input, with the option to halt calculation.

CART decision tree as output

Build a binary tree by iteratively performing the following operations on each node, starting with the root node, by the training data set:

Step 1: Calculate the Gini index of the data set's existing features with the node's training data set to D. Now, D is split into two halves, D1 and D2, and the formula $Gini(D, A)$ is used to determine the Gini index when each characteristic A and each potential value and are tested to see if they are "yes" or "no" at the sample point $A=a$.

Step 2: Choose the feature with the lower Gini index and its accompanying potential segmentation points as the most distinctive and effective segmentation points among all feasible characteristics A and all their potential segmentation points a. The training data set is allocated to the two child nodes by the optimal feature and the greatest split point, which are both derived from the current node.

Step 3: Recursively call Steps 1 and 2 on the two child nodes until the condition is satisfied.

Step 4: The number of samples in the node must be less than the predefined threshold, the sample set's Gini index must be less than the predetermined threshold, or there must be no more features for the CART decision tree method to halt.

B. Advantages and disadvantages of the decision tree algorithm

- (1) The generated decision tree is straightforward and easy to understand; there is essentially no need for preprocessing, to normalize or deal with missing values beforehand; it can handle both discrete and continuous values, whereas many algorithms only concentrate on discrete or continuous values value.
- (2) Unlike black-box classification models like neural networks, such as multi-dimensional output classification problems, the decision tree approach may be stated rationally.
- (3) The model can be chosen using cross-validation pruning to increase generalization ability; it has strong resilience and good fault tolerance for aberrant points.
- (4) The decision tree approach has a weak generalization capability since it is very simple to overfit. It can be improved by restricting the depth of the decision tree and the minimum number of node samples; otherwise, a slight change in the sample can cause the decision tree to drastically alter the tree structure, which can be resolved using techniques like ensemble learning.
- (5) The produced decision tree is likely to be biased towards specific features if the sample ratio of those features is too high. The sample weights can be changed to help with this.

IV. EXPERIMENTAL DESIGN

This study implements the recognition of handwritten digits by KNN classifier and decision tree classifier using Python language and sklearn library. The four key stages of implementation are: preparing the MNIST standard data set; handwriting recognition using KNN and decision trees; and evaluation of the final two classification models.

A. Standard dataset MNIST preprocessing

Figure 2 illustrates how to acquire the MNIST dataset used in this study from the official website. There are four files after downloading, including train-labels-idx 1-ubyte.gz, t10k-images-idx 3-ubyte.gz, train-labels-idx 1-ubyte.gz, and t10k-labels-idx 1-ubyte -gz are three examples. The training set image set, the training set image label set, the test set image set, and the test set image label set are all contained in these four files. Because the downloaded image data sets and labels are binary data streams saved in a unique format rather than images and text, these binary data streams must be processed to produce image and text files for machine learning.

THE MNIST DATABASE of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

Four files are available on this site:

```
train-images-10x10-bytestraining set images (992482 bytes)
train-images-10x10-jpegs10x10 training set images (29813 bytes)
train-images-10x10-zips10x10 training set images (1448577 bytes)
train-images-10x10-jpegs10x10 training set images (4548 bytes)
```

please note that your browser may uncompress these files without telling you. If the files you downloaded have a larger size than the above, they have been uncompressed by your browser. Simply rename them to remove the .gz extension. Some people have asked me "my application can't open your image files". These files are not in any standard image format. You have to write your own (very simple) program to read them. The file format is described at the bottom of this page.

The original black and white (bilevel) images from NIST were size normalized to fit in a 28x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. The images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

With some classification methods (particularly template-based methods, such as SVM and K-nearest neighbors), the error rate improves when the digits are centered by bounding box rather than center of mass. If you do this kind of pre-processing, you should report it in your publications.

The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Therefore it was necessary to build a new database by mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1. The 60,000 pattern training set contained examples from approximately 250 writers. We made sure that the sets of writers of the training set and test set were disjoint.

SD-1 contains 58,527 digit images written by 500 different writers. In contrast to SD-3, where blocks of data from each writer appeared in sequence, the data in SD-1 is scrambled. Writer identities for SD-1 is available and we used this information to unscramble the writers. We then split SD-1 in two: characters written by the first 250 writers went into our new training set. The remaining 250 writers were placed in our test set. Thus we had two sets with nearly 30,000 examples each. The new training set was completed with enough examples from SD-3, starting at pattern # 0, to make a full set of 60,000 training patterns. Similarly, the new test set was completed with SD-3 examples starting at pattern # 35,000 to make a full set with 60,000 test patterns. Only a subset of 10,000 test

Fig. 2. MNIST dataset official website

The initial data collection should be preprocessed and converted into a.csv file. Each row in the.csv file includes 784 cells, which correspond to each handwritten number because each one is 28 by 28. Due to the 60,000 images in the training set, there are 60,000 lines in the.csv file indicating all the numbers in the training set for each pixel of the image. To make it easier for later training and testing, the 60,000 photographs are split into 42,000 training pictures and 18,000 test pictures in a 7:3 ratio.

Figure 3 displays the visual statistics of these 60,000 images. Since each category's distribution of numbers is rather uniform, no additional preprocessing is necessary.

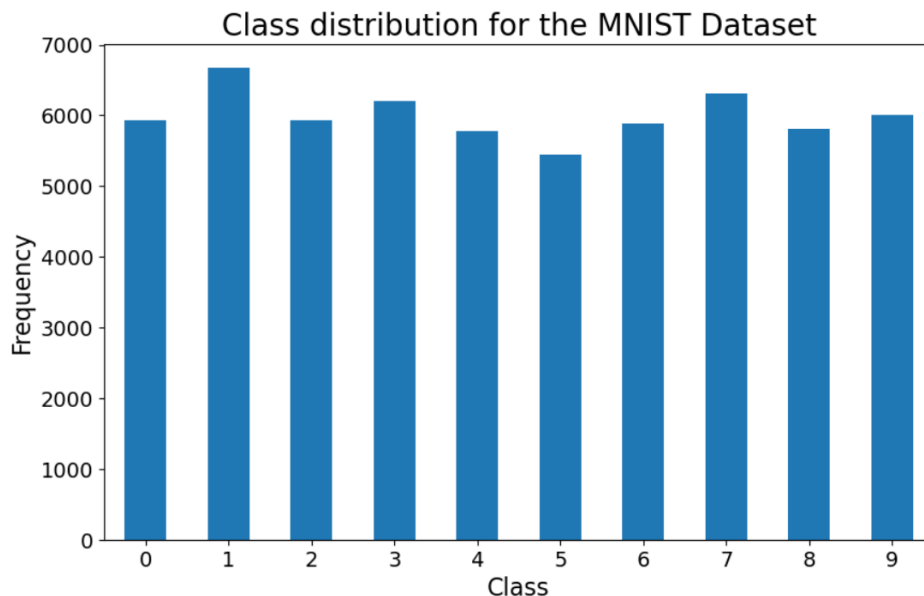


Fig. 3. Class distribution for the MNIST Dataset

B. KNN recognition of MNIST

Utilize the KNeighborsClassifier function of the Sklearn package to build a KNN model for training and prediction after processing the MNIST dataset. The KNeighborsClassifier function's K value must be set, and the ideal K value must be found. Build a KNN classifier for training and prediction, set the K value to a certain value in [2,20], and output the prediction accuracy corresponding to various K values simultaneously. Figure 4 depicts the output accuracy line graph.

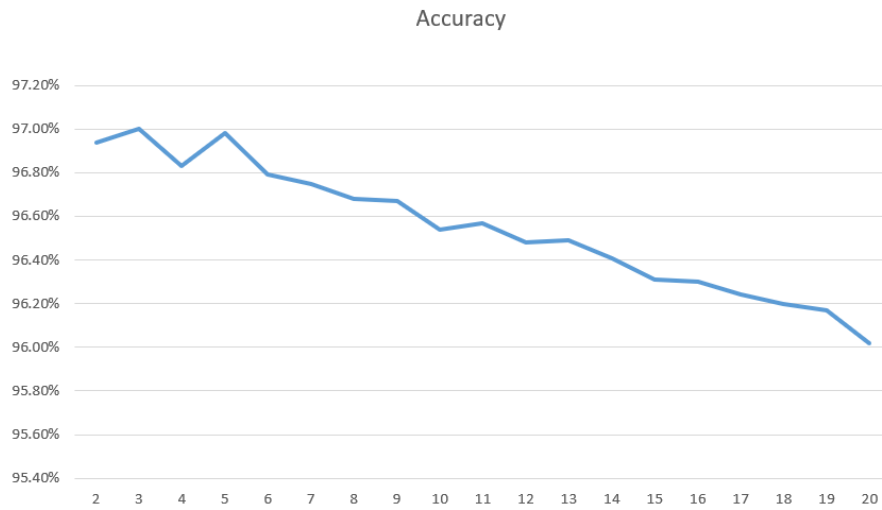


Fig. 4. Prediction accuracy corresponding to different K values

As can be seen, there is a modest peak in the forecast accuracy rate at K=3, and as K increases, there is a declining trend in the prediction accuracy rate. K=3 should therefore be the ideal K value. Create a KNN classifier with the obtained K value and train it using the training data. Utilize the model to predict the test set after it has been trained, then compare the predictions to the actual label value. The KNN classifier's predicted outcomes are as follows:

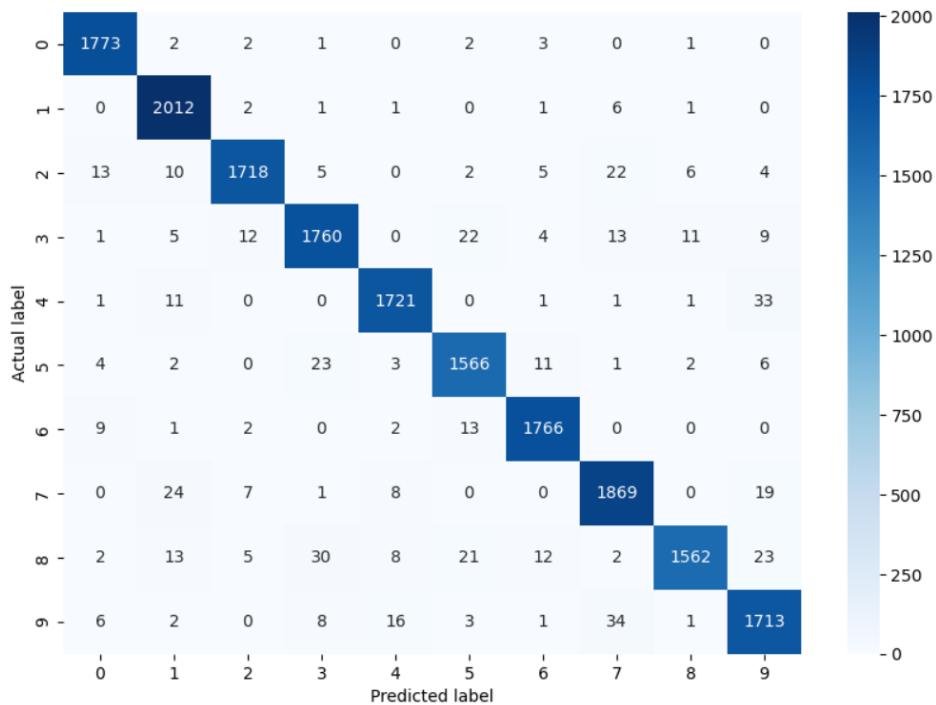


Fig. 5. Confusion matrix of KNN classifier prediction results

The classification of 9 and 7 provide the classifier with somewhat significant issues, as seen by the confusion matrix of the KNN classifier's prediction results in the above image.

Table. 1. KNN classifier prediction results in the evaluation table

	Precision	Recall	F1-score	Support
0	0.98	0.99	0.99	1784
1	0.97	0.99	0.98	2024
2	0.98	0.96	0.97	1785

3	0.96	0.96	0.96	1837
4	0.98	0.97	0.98	1769
5	0.96	0.97	0.96	1618
6	0.98	0.98	0.98	1793
7	0.96	0.97	0.96	1928
8	0.99	0.93	0.96	1678
9	0.95	0.96	0.95	1784
Accuracy			0.97	18000
Macro avg	0.97	0.97	0.97	18000
Weighted avg	0.97	0.97	0.97	18000

The evaluation table shows that the KNN classifier has an overall recognition accuracy rate of 97% on the MNIST test set. The recognition accuracy rates of numbers 0, 2, 4, 6, and 8 are all higher than the average, reaching 98% accuracy; in contrast, the recognition accuracy rates of numbers 3, 5, 7, and 9 are all lower than the average, with number 9 having only 95% accuracy. The recall rate for the number 0 is the highest, while the recall rate for the number 8 is the lowest.

C. Identification of MNIST by decision tree

Use the DecisionTreeClassifier function of the sklearn package to build a decision tree model for training and prediction after processing the MNIST dataset. To use the DecisionTreeClassifier function, many parameters must be set. Different decision tree parameters must be established to achieve the ideal parameter setting, and the ideal parameters can be discovered through training and result in feedback prediction. Following testing, the following decision tree parameters are chosen: The remaining values can be set to default using splitter="best", min samples split=2, and min samples leaf=1.

Create a decision tree classifier using the best parameters, then train it using the training data. Use the model to forecast the test set after it has been trained, then contrast the results with the actual label value. The following are the decision tree classifier's predictions:

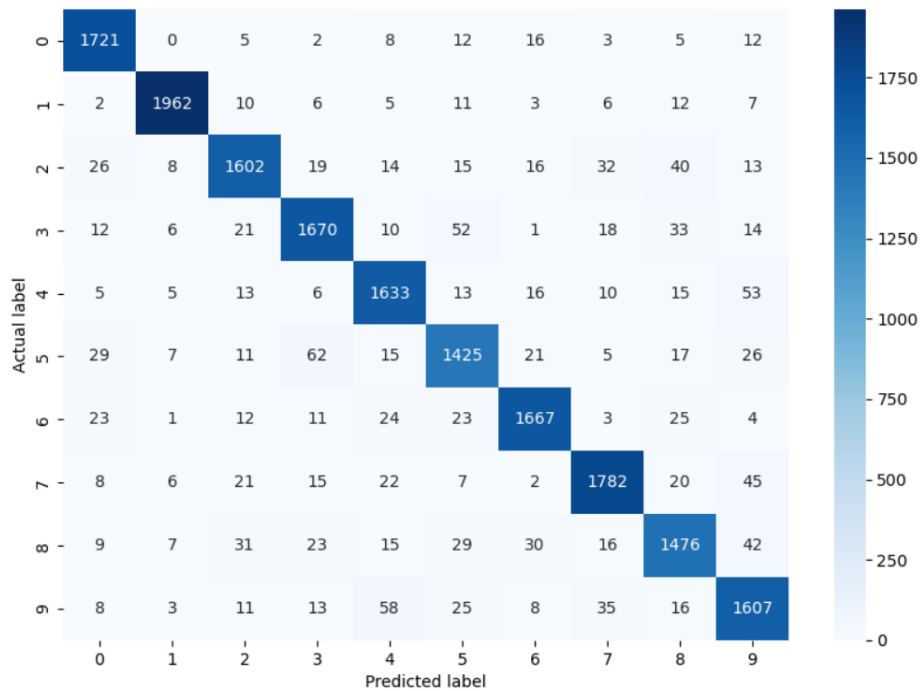


Fig. 6. Confusion matrix of prediction results of a decision tree classifier

It is clear from the confusion matrix of the decision classifier's prediction results in the aforementioned figure that the classifier has a significant issue correctly classifying the numbers 4 and 9, and that other numbers like 5 and 3 are also seriously misclassified.

Table. 2. Evaluation table of prediction results of a decision tree classifier

	Precision	Recall	F1-score	Support
0	0.93	0.96	0.95	1784
1	0.98	0.97	0.97	2024
2	0.92	0.90	0.91	1785
3	0.91	0.91	0.91	1837
4	0.91	0.92	0.91	1769
5	0.88	0.88	0.88	1618
6	0.94	0.93	0.93	1793
7	0.93	0.92	0.93	1928
8	0.89	0.88	0.88	1678
9	0.88	0.90	0.89	1784
Accuracy			0.97	18000
Macro avg	0.92	0.92	0.92	18000
Weighted avg	0.92	0.92	0.92	18000

The decision tree classifier's overall recognition accuracy rate on the MNIST test set is 92%, as can be seen from the assessment table. The recognition accuracy rates of numbers 0 and 6 are higher than the average, at 93%. Particularly for the digits 9 and 5, which had an accuracy rate of 88 percent, the recognition rate was below average. The recall rate for number one is the highest, while recall rates for numbers 8 and 5 are the lowest.

D. Comparison between the KNN classifier and decision tree classifier

The two experiment groups mentioned above lead to the following conclusion: (1) The confusion matrix demonstrates that the KNN classifier performs better on the MNIST dataset; (2) the evaluation table demonstrates that the KNN classifier performs better in each digital category. The final accuracy rate is 97%, which is likewise greater than the decision tree classifier's accuracy rate of 92%. The precision, recall, and F1-score on the above are all higher than the choice tree classifier.

V. CONCLUSION

The KNN and decision tree classifier are implemented in this study using the Python and Sklearn libraries. Their accuracy is tested using the MNIST standard dataset for handwritten digit images. This method first separates the image binary stream and label binary stream in the MNIST dataset to obtain the image and label data for training and testing, then determines the best KNN and decision tree parameters through testing, and finally constructs the KNN classifier and decision tree classification, using the divided training set to train the model and the model to test. Following testing and comparison, the KNN classifier's accuracy reached 97% and it outperformed the decision tree classifier in metrics like Precision, Recall, and F1-score, showing that the KNN algorithm is more adept at identifying handwritten numbers and that the KNN classification device model is superior.

REFERENCES

- [1] Li Zhiwei. An Overview of Pattern Recognition System and Statistical Pattern Recognition Research[J]. Examination Weekly, 2016(88):106.
- [2] Ahlawat S, Choudhary A, Nayyar A, et al. Improved handwritten digit recognition using convolutional neural networks (CNN)[J]. Sensors, 2020, 20(12): 3344.
- [3] Deng L. The mnist database of handwritten digit images for machine learning research [best of the web][J]. IEEE signal processing magazine, 2012, 29(6): 141-142.
- [4] Cover T, Hart P. Nearest neighbor pattern classification[J]. IEEE transactions on information theory, 1967, 13(1): 21-27.
- [5] Quinlan J R. Learning decision tree classifiers[J]. ACM Computing Surveys (CSUR), 1996, 28(1): 71-72.