Southern University of Science and Technology
Department of Mechanical and Energy Engineering
Student name: Hanqi Su, Dihan Liu, Zongze Li
Student ID: 11912625 11912532 11911511

Semester 1, 2022
Midterm Project - MNIST

ML Project - MNIST Handwritten Digital Image Classification

1 Introduction of mnist and our work

1.1 Introduction of mnist

Minist data set is a very classic data set in the field of machine learning, consisting of 0-9 handwritten digital images and digital labels, similar to the status of "Hello World" in programming. The data provided on the official website are four compressed files: training picture, training picture label, test picture and test picture label. There are 60,000 data in the training set and 10,000 data in the test set. Each sample is a 28*28 pixel gray handwritten digital image. In this mid-term project, in addition to the data set on the official website, we also used the handwritten number set provided by the class teacher team, including 2,840 training samples and 900 test samples.

1.2 Introduction of our work

In this mid-term project, our team first used image optimization algorithm, PCA and cross-validation to process the data set, and then used Logistic regression, Naive Bayes Classifier, SVM with Different Kernels, Random Forest, KNN, K-means ,MLP and other algorithms realize the binary classification of the number 35 and 89 in the digital data set and the multi-classification task of the number 4567, then we adjust some parameters in the algorithm to obtain higher operation speed and accuracy. Finally, the confusion matrix of classification results is drawn to show the results in a visual way, which successfully completes the requirements of the mid-term project.

2 Introduction to model and specific function

2.1 Logistic Regression algorithm

Sigmoid function

$$S_i(x) = \frac{1}{1 + e^{-z_i}} \tag{1}$$

Softmax function

$$S_i(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

In most cases, the logistic method is used to classify problems using sigmoid and Softmax, and there is no difference in the mathematical expressions of the two in solving dichotomies. Since sigmoid is used to solve dichotomous problems in most cases, it is troublesome to repeatedly dichotomize multiple objects when solving multi-classification problems. Softmax function can not only solve the problem of binary classification but also directly used for multi-classification problems, wide applicability and convenient use. So softmax function is used when logistic method is used to classify handwritten numbers.

Then cross entropy is used as the loss function:

$$L_i(W,b) = -log(S_{yi}(Wx_i + b))$$

The gradient descent method is used to optimize the iteration of the loss function, and the gradient expression of the loss function is as follows:

$$\frac{\partial L_i}{\partial W_{ij}} = \begin{cases} (S_i - 1)x_j & , y = i \\ S_i x_j & , y \neq i \end{cases} \tag{2}$$

$$\frac{\partial L_i}{\partial b_i} = \begin{cases} (S_i - 1) & , y = i \\ S_i & , y \neq i \end{cases} \tag{3}$$

The general running process of the whole algorithm is as follows. The label of the image can be transformed into a 10*1 unit vector as yi (the corresponding value of the label in the figure is 0), and the image can be transformed into a 784*1 vector as XI. By using W and B, the result of Wxi +b can be as close to YI as possible, and the approximation degree is described by the cross entropy loss function. The optimization process of the loss function is realized by the gradient descent method mentioned above. By continuously introducing Xi and Yi to optimize the iteration of W and B, the model training is finally completed and the optimal W * and B * are obtained. For any test sample X, w * and B * can be used to obtain a predicted Y, and the number of pins of the element with the largest value in the vector minus 1 is the predicted number of the image.

$$Wx_i + b = \hat{y} \sim y_i \ (m = 10, n = 784) \tag{4}$$

$$\begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} f_1(w, x_i, b) \\ f_2(w, x_i, b) \\ \vdots \\ f_m(w, x_i, b) \end{bmatrix} \sim \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Finally, the parameter training of W and b is completed by constantly bringing in training data. The handwritten cyclic iteration code is as follows:

```
def train_function(batch_size, alpha, training_times):
    w = np.zeros(w_dim)
    b = np.zeros(b_dim)
    x_batches = np.zeros(((int(x_train_use.shape[0] / batch_size), batch_size, 784)))
    y_batches = np.zeros(((int(x_train_use.shape[0] / batch_size), batch_size)))
    batches_num = int(x_train_use.shape[0] / batch_size)
    for i in range(0, x_train_use.shape[0], batch_size):
        x_batches[int(i / batch_size)] = x_train_use[i:i + batch_size]
        y_batches[int(i / batch_size)] = y_train_use[i:i + batch_size]
    print('Start training...')

    for times in range(training_times):
        for i in range(batches_num):
            w_gradients = np.zeros(w_dim)
            b_gradients = np.zeros(b_dim)
            x_batch = x_batches[i]
            y_batch = y_batches[i]
            for j in range(batch_size):
                w_g, b_g = loss_function_gradient(w, b, x_batch[j], y_batch[j])
                w_gradients += w_g
                b_gradients += b_g
            w_gradients /= batch_size
            b_gradients /= batch_size
            w -= alpha * w_gradients
            b -= alpha * b_gradients
    return w, b
```

Figure 1: Code of training W and b

2.2 Naive Bayes

This algorithm combines prior probability of a class with new information, using new data to update conditional probabilities and gradually improve accuracy. We have learnt in class that the features in Bayes classifier are independent. Take y as target and x as features. We have formula (5)

$$P(y|X) = \frac{P(y) * P(X|y)}{P(X)} = \frac{P(y)}{P(X)} \prod_{i=1}^{d} P(x_i|y) \tag{5}$$

It is equivalent to

$$max \prod_{i=1}^{d} P(y)P(x_i|y)$$

Take loglikelihood to simplify calculation

$$maxlog[P(y)] + \sum_{i=1}^{d} log[P(x_i|y)] \prod_{i=1}^{d} P(y)P(x_i|y)$$

Some times specific classes may not appear(Although it will not appear in our project), $P(y)$ and $P(X|y)$ may be 0. Calculate $P(y) * P(X|y)$ has a value of 0. It loose the meaning to compare and not easy to derive log likelihood, so we need to "smooth" the probability value, commonly used Laplace correction which add 1 to every class. Let A_y represent the set of sample belongs to class y in training set A, and N represent the number of possible classes in training set A. We have function

$$P(y) = \frac{|A_y| + 1}{|A| + N} \tag{6}$$

Specifically, in our code: We take every pixel as a feature and count prior probability and conditional probability from the training set. Function $Train()$ gets these information. First, build prior probability array in $1 * classes$ and conditional probability array in $classes * feature(inthisproject784) * 2$. The "2" refer to probability equals 0 or 1. We only need to know if the feature emerged, so binarize every pixel to (0,1). After that, prior probability is simply derived through counting label number and conditional probability count the number of 0 or 1 looping though all features. Importantly, we need to map the probability to 1.10001 which is the same effect as the smooth we mentioned before. Function Predict dose prediction for testing set with return of prior and conditional probability from function $Train()$. At last, accuracy is derived by comparing labels returned by function $Predict()$ and actual labels.

2.3 SVM and Kernel Method

SVM refer to support vector machine. Support vector machine is a binary classification model. Its basic model is a linear classifier defined in feature space with the largest interval, which distinguishes it from perceptron. SVM also includes nuclear tricks, which makes it a substantially nonlinear classifier. The learning strategy of SVM is interval maximization, which can be formalized as a problem of solving convex quadratic programming, and is equivalent to the minimization problem of regularized hinges loss function.

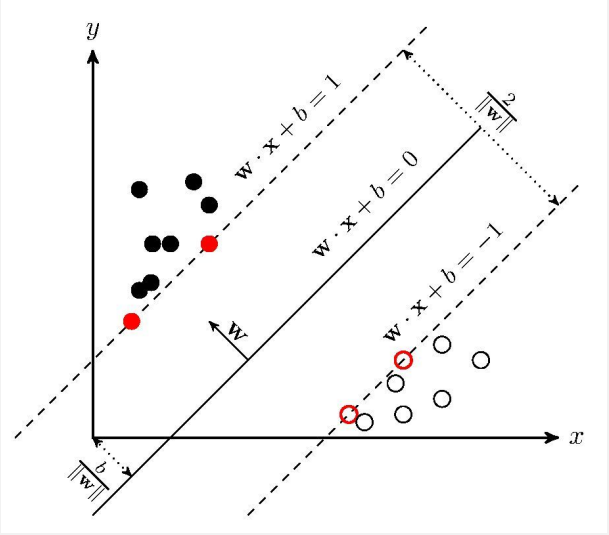


Figure 2: Code of normalization

The basic idea of SVM learning is to solve the separated hyperplane which can divide the training data set correctly and has the maximum geometric spacing. As shown in the figure, $wx + b = 0$ is the separated hyperplane. For a linearly separable data set, there are infinitely many such hyperplanes (i.e. perceptrons), but the separated hyperplane with the largest geometric interval is unique. Maximum hyperplane requirements: the two types of samples are segmented on both sides of the hyperplane. The distance between the sample points closest to the hyperplane on both sides is maximized. In n dimension, we have formular for hyperplane:

$$w^T x + b = 0 \tag{7}$$

The distance of point $x = (x_1, x_2, ..., x_n)$ to the line is:

$$d = \frac{|w^T x + b|}{||w||}$$

According to the definition of the support vector, the distance between the support vector and the hyperplane is d, and the distance between other points and the hyperplane is greater than d. We have equation (8)

$$\begin{cases} \dfrac{|w^T x + b|}{||w||} \geq d & y = 1 \\ \dfrac{|w^T x + b|}{||w||} \leq -d & y = -1 \end{cases} \tag{8}$$

This is equal to (9)

$$y(w^T x + b) \geq 1 \tag{9}$$

Maximize d with formula, we can get the optimized problem.

$$min \frac{1}{2}||w||^2 \ s.t. \ \ y_i(w^T x_i + b) \geq 1$$

We construct Lagrange function for inequality constrained optimization

$$\min_{w,b} \ \max_{\lambda} \ L(w, b, \lambda) = \frac{1}{2}||w||^2 + \sum_{i=1}^{n} \lambda_i [1 - y_i(w^T x_i + b)]$$

$$s.t. \ \ \lambda_i \geq 0$$

Transform using strong duality:

$$\max_{\lambda} \ \min_{w,b} \ L(w, b, \lambda)$$

After calculation, we get

$$\max_{\lambda}[\sum_{j=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (x_i x_j)] \tag{10}$$

$$s.t. \ \ \sum_{i=1}^{n} \lambda_i y_i = 0 \ \ \lambda_i \geq 0$$

In practical application, there are very few samples that are completely linearly separable. In case of samples that are not completely linearly separable, soft intervals can be added to allow individual samples to appear in the interval band. Introduce a relaxation variable ξ for each sample and they meets the constraints.By increasing the soft interval our optimization goal becomes:

$$\min_{w} \frac{1}{2}||w||^2 + C \sum_{i=1}^{m} \xi_i \tag{11}$$

$$s.t. \ \ g_i(w, b) = 1 - y_i(w^T x_i + b) - \xi_i \leq 0, \ \ \xi_i \geq 0, \ \ i = 1, 2, ..., n$$

Where C is a constant greater than 0, which can be understood as the degree of punishment for the wrong sample.

The hard and soft intervals of the sample points I just mentioned are all cases where the sample is completely or mostly linearly separable.But there may be cases where the sample points are not linearly separable. For the linearly indivisible samples in the finite dimensional vector space, we map them to the vector space of higher dimensions, and then learn the support vector machine by the way of spacing maximization, which is nonlinear SVM. We use x to represent the original sample point, and we use $\phi(x)$ to represent x mapping to the new feature space to the new vector. Then the segmented hyperplane can be expressed as: $f(x) = w^T \phi(x) + b$

The dual problem of nonlinear SVM becomes:

$$\min_{\lambda}[\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (\phi(x_i) \cdot \phi(x_j)) - \sum_{j=1}^{n} \lambda_i] \tag{12}$$

$$s.t. \ \ \sum_{i=1}^{n} \lambda_i y_i = 0 \ \ \lambda_i \geq 0, \ \ C - \lambda_i - \mu_i = 0$$

Kernel method

After low-dimensional space is mapped to high-dimensional space, the dimension may be very large, and the dot product of all samples is too much to calculate. Using the kernel function$k(x, y) = (\phi(x), \phi(y))$, the inner product of x_i and x_j in the feature space is equal to their result computed by the function $k(x, y)$ in the original sample space. The introduction of the visible kernel reduces the amount of computation on the one hand and the amount of memory used to store data on the other.

Commonly used kernel functions are:

$$Linear \ kernel \ k(x_i, x_j) = x_i^T x_j$$
$$Polynomial \ kernel \ k(x_i, x_j) = (x_i^T x_j)^d$$
$$Gaussian \ kernel \ k(x_i, x_j) = exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$$
$$Laplace \ kernel \ k(x_i, x_j) = exp(-\frac{||x_i - x_j||}{\sigma})$$
$$Sigmoid \ kernel \ k(x_i, x_j) = tanh(\beta x_i^T x_j + \theta)$$

Specifically in the code, we used package of sklearn. For SVM we use function LinearSVC function for linear model. Function sim.fit is used to train the data and find the best hyperplane. Function svm.predict is used to get the testing label. When practicing kernel method to make predictions, we apply polynomial kernel and rbf kernel. Funtion $SVC(kernel =' poly', C = 1, degree = 3)$ is used to test polynomial kernel. Function $SVC(kernel =' rbf', C = 100, gamma = 0.01)$is used to test rbf kernel.

2.4 Random Forest

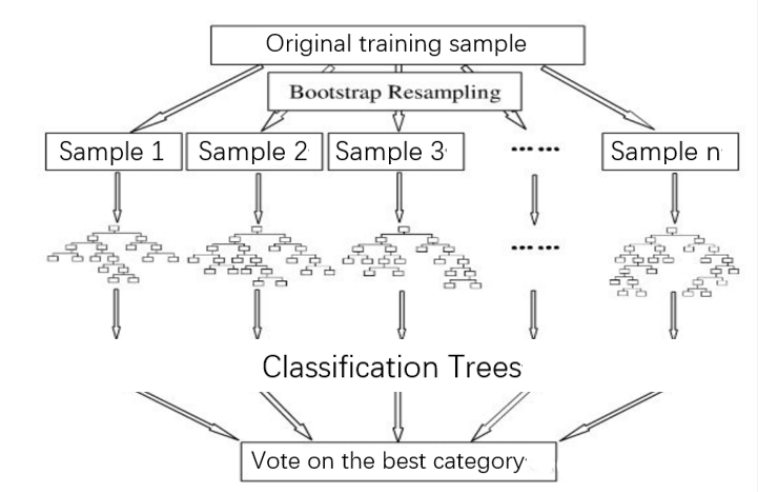


Figure 3: Flow chart of random forest algorithm

Random forest is to generate a new training sample set by repeatedly and randomly selecting K samples from the original training sample set N, and then generating k classification trees according to the self-service sample set to form a random forest. The classification result of new data is determined by the score formed by the number of votes in the classification tree. In essence, it is an improvement of decision tree algorithm, which merges multiple decision trees together, and the establishment of each tree depends on an independent sample extraction. Each tree in the forest has the same distribution, and the classification error depends on the classification ability of each tree and the correlation between them. Feature selection uses a random method to split each node, and then compares the errors generated in different cases. The intrinsic estimation error that can be detected, the ability to classify, and the correlation determine the number of selected features. The classification ability of a single tree may be small, but after a large number of decision trees are randomly generated, a test sample can statistically select the most likely classification through the classification results of each tree. In the process of building each decision tree, two points need to be noted: sampling and complete splitting. Firstly, there are two random sampling processes. Random Forest samples the input data in rows and columns. For row sampling, the method of putting back is adopted, that is, there may be duplicate samples in the sample set obtained by sampling. If we take N samples, we take N samples. In this way, the input samples of each tree are not all samples during training, making over-fitting relatively difficult. Then, column sampling is carried out, and M features. Then, a decision tree is established by completely splitting the sampled data. In this way, a certain leaf node of the decision tree either cannot continue splitting, or all the samples in the tree point to the same classification.

Bagging is the core strategy of random forest, which firstly generates different training sets through self-sampling method, and then trains the base learner based on this new training set. Finally, the prediction result of ensemble learner will be obtained by k base learner generated after k round iteration through simple voting method. Its mathematical expression is as follows:

Training set $\boldsymbol{D} = [(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)]$, Decision tree (Base learner) algorithm: $\boldsymbol{\zeta}$, Training times: K.

1.for t=1,2,...K do 2.$h_t = \zeta(D, D_{bs})$ 3.end for

output:

$$H(x) = arg\ max \sum_{t=1}^{K} (h_t(x) = y) \tag{13}$$

In short, there are many decision trees in the random forest. Different training sets are used to train different decision trees, and test sets are used to test the decision trees. The final output results are decided by voting results obtained by all decision trees.

2.5 KNN

KNN is a kind of supervised learning and stands for k nearest neighbors. The algorithm compares the distance of training samples and testing samples, then extract k nearest training sample data to make statistics on the classification labels of the K training sample data, among which the category represented by the label with the nearest is the category of the data to be tested. The distance calculate formula is (14)

$$L_p(x_i, x_j) = \left(\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|^p \right)^{\frac{1}{p}} \tag{14}$$

When p = 1, it represents Manhattan distance.
When p = 2, it represents Euclidean distance.
And we usually use Euclidean distance for calculation.
When k is bigger than 1, there are lot of distance to be considered. But sometimes, some sample may be really far away from the testing sample. That training sample is meaningless to consider. So according to distance we can give a weight to each of the neighbors. The weight can be $\frac{1}{distance}$
Specifically, when we derive our code. We use knn package in sklearn. $KNeighborsClassifier()$ is used to define k, $knn.fit()$ to train the data, and $knn.score()$ to predict and get accuracy. Doing data processing at the beginning, then test the best k for training set, test the bet k and methods for testing set. In funtion $KNeighborsClassifier()$ there are methods "uniform" and "distance", "distance" method is what we mentioned to give weight for neighbors. From the program we can get the best k, method, accuracy and time.

2.6 K-means

K-means is a kind of unsupervised learning, information of labels in unused in the algorithm. People give the cluster number and the machine will divide the samples though distance to the center. It starts with randomly giving centers which is the same amount with clusters. Next, calculate the distance of centers to samples. Samples belongs to the cluster with nearest distance. The calculate formula in knn is also used hear. Moreover, sometimes the cosine distance(formula(15)) is used.

$$cos\theta = \frac{A \cdot B}{|A||B|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{15}$$

After dividing, with samples in new clusters, calculate the mean distance and that represents the new center of clusters. With new samples to train, the algorithm goes a loop for calculate sample distance with centers–new samples classified to the nearest clusters–update centers–calculate distance... It stops if
(1)the centers is not changing
(2)iterations reach the maximum iterations
(3)sum of mean square error smaller than a specific value

For predicting, simply calculate the distance with every centers, and the test sample belongs to the nearest center.
Specifically, in our code. We reconstruct this algorithm by ourselves. First we give a class named Kmeans, it is convenient to initialize. Kmeans needs parameters k, data, labels, randommatrix. Runnnig function $work()$ will start to train the data. In this function it will run $get_centre()$ first which is used to get random matrix for centers. If parameter randommatrix is an integer, matrix will be given randomly, randommatrix is a matrix, initialize will use the matrix as a fake randommatrix. This is for optimizing accuracy and will be used later. Funtion $update\_centre()$ is calculation new centers with function $means()$ of classification. In function $get\_distances(self)$ we calculate distance using Euclidean method and save them to a one dimension array. Function $classify()$ classifies the new sample to clusters, it will get the index of the minimum value of each row in the distance array, save as cluster labels. Function $update\_centre()$, $get\_distances(self)$ and $classify()$ can form a loop in maximum iterations.
For prediction we only need centers and calculate the distance, so these processes are packaged in to function $test()$.

2.7 MLP

MLP refers to multilayer perceptron.The most typical MLP consists of three layers: input layer, hidden layer and output layer, any neuron in the layer above has connections to all the neurons in the layer below.

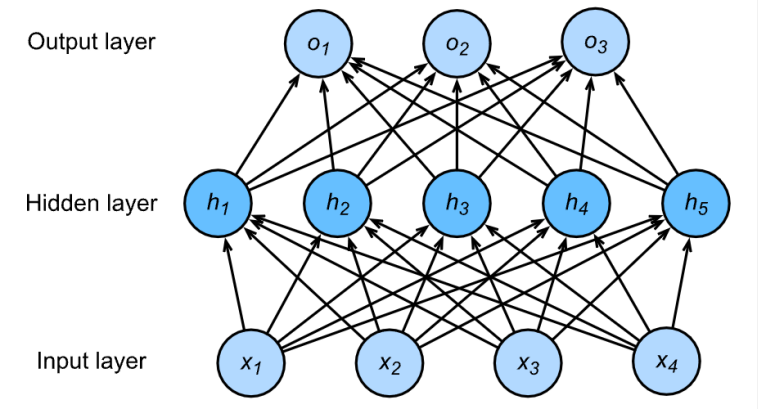


Figure 4: Layer graphs of MLP

Given a small batch sample $X \in \mathbb{R}^{n\times d}$,the batch size is n, and the input number is d. Suppose that the multilayer perceptron has only one hidden layer, where the number of hidden units is h.The output of the hidden layer (also known as the hidden layer variable or hidden variable) is H, and we have $H \in \mathbb{R}^{n\times H}$. Since both the hidden layer and the output layer are fully connected layers, the weight parameters and deviation parameters of the hidden layer can be set as $W_h \in \mathbb{R}^{d\times h}$ and $b_h \in \mathbb{R}^{1\times h}$ respectively, the weight and deviation parameters of the output layer are $W_o \in \mathbb{R}^{h\times q}$ and $b_o \in \mathbb{R}^{1\times q}$ respectively.
Take the output of the hidden layer directly as the input of the output layer, its output $O \in \mathbb{R}^{n\times q}$ is calculated as

$$W = XW_h + b_h \tag{16}$$

$$O = HW_o + b_o \tag{17}$$

Combine equations(16)(17) we can get:

$$\begin{aligned} O &= (XW_h + b_o)W_o + b_o \\ &= XW_hW_o + b_hW_o + b_o \end{aligned}$$

The output layer weight parameter is $W_h W_o$, the deviation parameter is $b_h W_o$. The full connection layer only performs affine transformation on the data, so we need to use the activation function to introduce nonlinear factors into the neuron, so that the neural network can approach any nonlinear function arbitrarily, so that the neural network can be used in more nonlinear models. The activation function should be continuous and differentiable, and the activation function and its derivative should be as simple as possible. The range of the derivative of the activation function should be within an appropriate interval, neither too large nor too small, otherwise it will affect the efficiency and stability of training. Usually it can be sigmoid or tanh function. Specifically, when we reproduce the algorithm, we not only the package in sklearn and but also write simple code by ourselves. In our code, we have feedforward() function to define the forward propagation function, get_gradient() function to calcaulate gradient and MLP can be successfully implemented.

3 Data processing

3.1 Load and get specific data

In the step of reading all the data, one of method we use the code provided by the teaching assistant as follows

```
f = gzip.open('test-me338-900-images-idx3-ubyte.gz', 'r')
image_size = 28
num_images = 900
f.read(16)
buf = f.read(image_size * image_size * num_images)
test_data = np.frombuffer(buf, dtype=np.uint8).astype(np.float32)
# print(test_data.shape)
test_data = test_data.reshape(num_images, image_size * image_size)
```

Figure 5: Data loading

Another way to read the code is to decompress the data file, convert it into a CSV file with specific programme(data2csv in our package), and then read the CSV file. Both methods get data in an N*784 matrix and labels in one demension.
According to the project requirements, We need to extract specific numbers of data. Here we use the method of traversing the array to examine the value of labels, if the label is what we need, the data will be stored in a new array.

```
y_train = np.uint8([])
x_train = np.ones((1, 784), dtype=np.uint8)
for loop in range(2840):
    if train_label[loop] == 4 or train_label[loop] == 5 or train_label[loop] == 6 or train_label[loop] == 7:
        y_train = np.append(y_train, train_label[loop].astype(np.uint8))
        x_train = np.append(x_train, np.expand_dims(train_data[loop], 0), axis=0)
        # a = 1
        # print(a)
x_train = np.delete(x_train, 0, 0)  # Delete the original first one matrix
# get test data: x_test  and  y_test
y_test = np.uint8([])
x_test = np.ones((1, 784), dtype=np.uint8)
for loop in range(900):
    if test_label[loop] == 4 or test_label[loop] == 5 or test_label[loop] == 6 or test_label[loop] == 7:
        y_test = np.append(y_test, test_label[loop].astype(np.uint8))
        x_test = np.append(x_test, np.expand_dims(test_data[loop], 0), axis=0)
x_test = np.delete(x_test, 0, 0)  # Delete the original first one matrix
```

Figure 6: Get specific data

The demension of data and labels keep the same as before.

3.2 Image optimization

After loading the specific data, we do image processing. Here we have two different algorithms to do image processing

3.2.1 SU algorithm

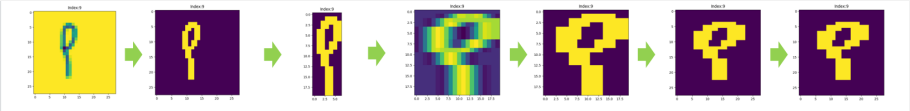


Figure 7: Process of SU algorithm

The core idea of this algorithm is to cut out the matrix according to the edges of the graph and stretch it to the size we want. Figure 1 shows the process of it. First we get the original data of training and testing sets in size of 28*28. Do binarization to the figure make it boundary clearer.We find the edge of the digital body and cut the smallest matrix that can accommodate the number.The method we use to find the edge of a number is to go through the array from top to bottom or left to right and find the last row or column that is not 0 to find the edge number.Reverse the operation and we can find the left and upper edge. Next we stretch the matrix to 20*20. We used interpolation function $np.real(cv2.resize(array2, dsize = (20, 20), interpolation = cv2.INTER\_CUBIC))$ in package of cv2. After completing this step, we can find that its boundary is a transition from the color middle to the surrounding. Next, we do binarization again to make the edges clearer.Since our target figure is 28×28 and the matrix we have know is much smaller, this 20 by 20 matrix will be placed in the center of the 28 by 28 matrix. In this way, the amount of information to be processed by the algorithm will be at a more appropriate level, and the display effect of the picture will also be better. And now we get a optimized 28*28 picture and save the information in matrix.

3.2.2 LIU algorithm

Figure 8: Process of LIU algorithm

When doing the project, this algorithm is aim to convert every written number to the same size and keep what it is like at the same time with the guessing that multiple form of numbers is better for machine learning and further tests. This algorithm have the similar logic with the former onefigure 1 shows the process of it. First we get the original data of training and testing sets in size of 28*28. Do binarization to the figure make it boundary clearer. After that, we are going to cut the picture. The logic is to get minimum number of each row and column. If it is not 255 than it means the component of figure is in the row or column. We can get the boundary of numbers through getting the start and stop row or column. With the boundary, we can determine the cutting picture size. Because this algorithm is cutting square matrix, we pick the longest side as length of square. To scaling the cut, we use Nearest Interpolation method for pictures and wrote the code by ourselves.

```
m = 28  # 求出变换后的坐标的最大值
n = 28
nn = m / Row  # 放大倍数
B = np.zeros([m, n])  # 定义变换后的矩阵
for i in range(m):
    for j in range(n):
        x = int(round(i/nn, 0))
        y = int(round(j/nn, 0))  # 最小临近法对图像进行插值
        if x == 0:
            x = 1
        if y == 0:
            y = 1
        if x >= Row:
            x = Row-1
        if y >= Col:
            y = Col-1
        B[i, j] = anew[x, y]
```

Figure 9: Code of Nearest Interpolation method

The main idea is to divide the index value with amplify rate and turn the result to integer. The color number of a pixel in the amplified matrix is the same with the pixel which the integer pointed to in the cut. We get a 28*28 picture and save the information in matrix.

3.3 Normalization

```
# 数据预处理
MTrain, NTrain = np.shape(x_train)  # 行列数
print("training: ", MTrain, NTrain)
xTrain = x_train[:, 0:NTrain]
xTrain_col_avg = np.mean(xTrain, axis=0)  # 对各列求均值
xTrain = (xTrain - xTrain_col_avg) / 255  # 归一化
print(xTrain)
x_train = xTrain

MTest, NTest = np.shape(x_test)  # 行列数
print("test: ", MTest, NTest)
xTest = x_test[:, 0:NTest]
xTest_col_avg = np.mean(xTest, axis=0)  # 对各列求均值
xTest = (xTest - xTest_col_avg) / 255  # 归一化
print(xTest)
x_test = xTest
```

Figure 10: Code of normalization

In order to eliminate the dimensional influence between indicators, data standardization is needed to solve the comparability between data indicators. Normalization aims to restrict the preprocessed data to a certain range (such as [0,1]), so as to eliminate the adverse effects caused by singular sample data.

Benefits:
(1)The normalization accelerates the speed of finding the optimal solution of gradient descent, improves the convergence speed of training network.
(2)It may improve the accuracy.

3.4 PCA

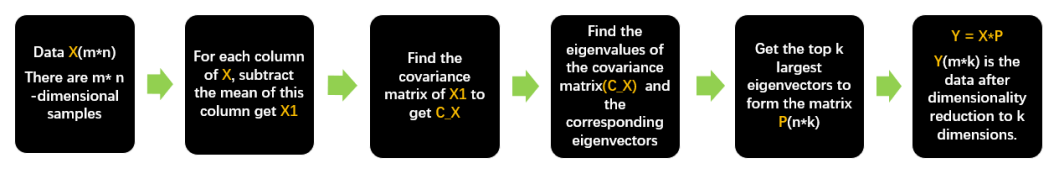


Figure 11: Process of PCA

PCA refers to Principal Component Analysis. The work of PCA is to find a set of mutually orthogonal coordinate axes sequentially from the original space. The selection of the new coordinate axes is closely related to the data itself. Among them, the first new coordinate axis is selected in the direction of the largest variance in the original data, the second new coordinate axis is selected in the plane orthogonal to the first coordinate axis to make the largest variance, and the third axis is selected in the plane orthogonal to the first and second axes to make the largest variance. And so on, you get k of these axes.

3.5 Monte Carlo cross validation

```
for i in range(n):
    x_train, x_test, y_train, y_test = train_test_split(x_dataset, y_dataset, test_size=0.1)
    ......
accurate = accurate_sum/n
```

Figure 12: Code of Monte Carlo cross validation

A data set is divided randomly repeatedly and combined into different training sets and test sets. The training set is used to train the model, and the test set is used to evaluate the model. Finally, the accuracy of each combination is averaged as the final accuracy.

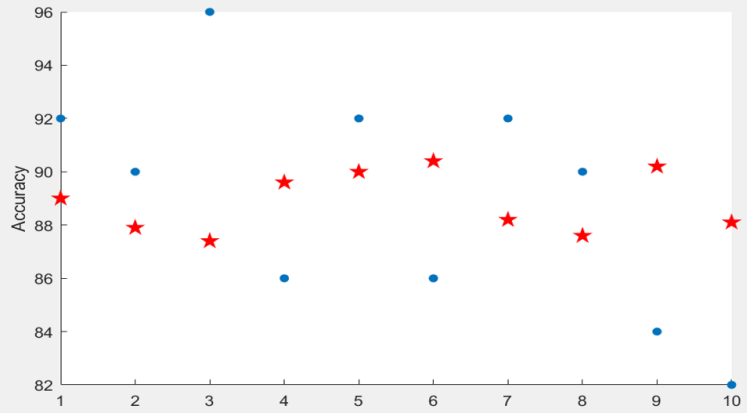


Figure 13: Comparison of results

In the case of no cross-validation, the test results of ten different training sets and test sets are shown in the blue circle in the figure, and the fluctuation of accuracy is relatively large. Therefore, if the division method of our training set and test set is not good enough, the best model and parameters may not be selected.

For the same data set, ten Monte Carlo cross validation was used, and the test results obtained were shown as the red five-pointed star in the figure. The mean accuracy was similar to that without cross validation, but the fluctuation of accuracy was significantly reduced. The accuracy is more stable

Advantages : Overfitting can be reduced to a certain extent. More information can be obtained from limited data. Stability of operation results can be improved.

Disadvantage: More data is used at run , which increases the running time.

Before:1.6296s After:14.6459s

4 Parameter tuning

4.1 SVM-C and SVM-penalty

In svm, The C value is the penalty factor or penalty factor as we metioned in the part Introduction of method, which describes the tolerance of the model for error. It's essentially a regularized coefficient. The higher C is, the less error is tolerated and the easier it is to overfit. The smaller C is, the less fit it is. If C is too large or too small, the generalization ability becomes worse.In order to show the role and function of parameter C and select the best title, we loop c from 10^{-6} to 10^{6} with intervals of multiples of 10 and get the accuracy. After drawing the image, we get the accuracy and put them in the figure below.

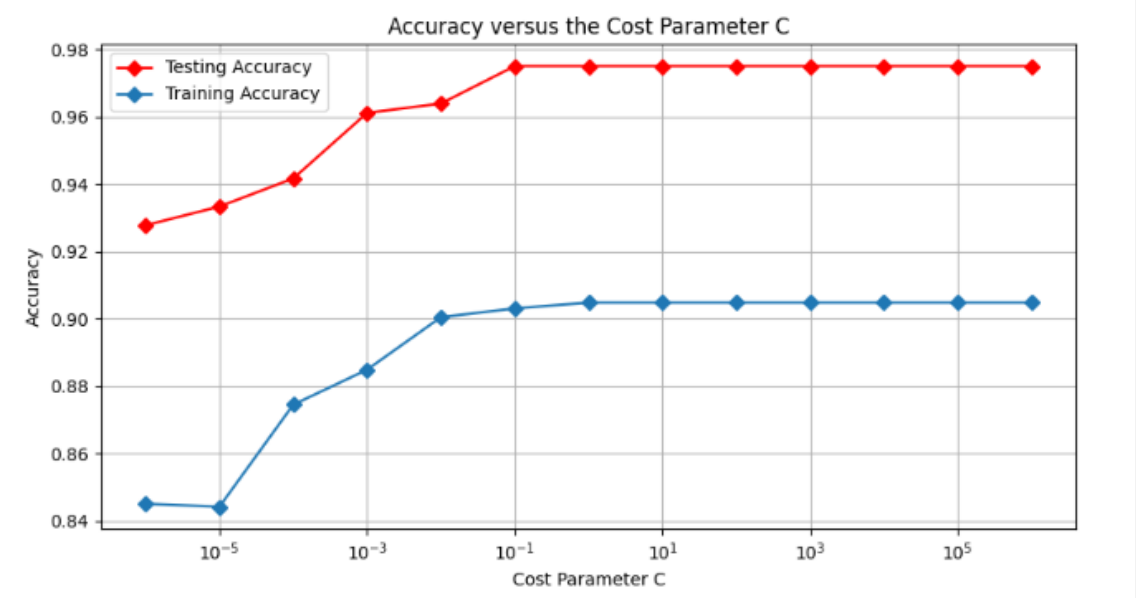


Figure 14: Parameter tuning of penalty c

From the figure, we can see that both the training accuracy and the test accuracy increase with the increase of C and tends to be stable at $C = 0.1$. We can also conclude that the most stable and appropriate value of C is 1 for our test sample.
In function *LinearSVC* there is another parameter which makes difference to the accuracy. Penalty parameter 'l1' of 'l2' can be chosen in the function. This parameter specifies the norm used in the penalization.
The 'l2' penalty is the standard used in SVC, effect is to is to smooth the weights. The 'l1' leads to 'coef_' vectors that are sparse. Results are shown in figure().

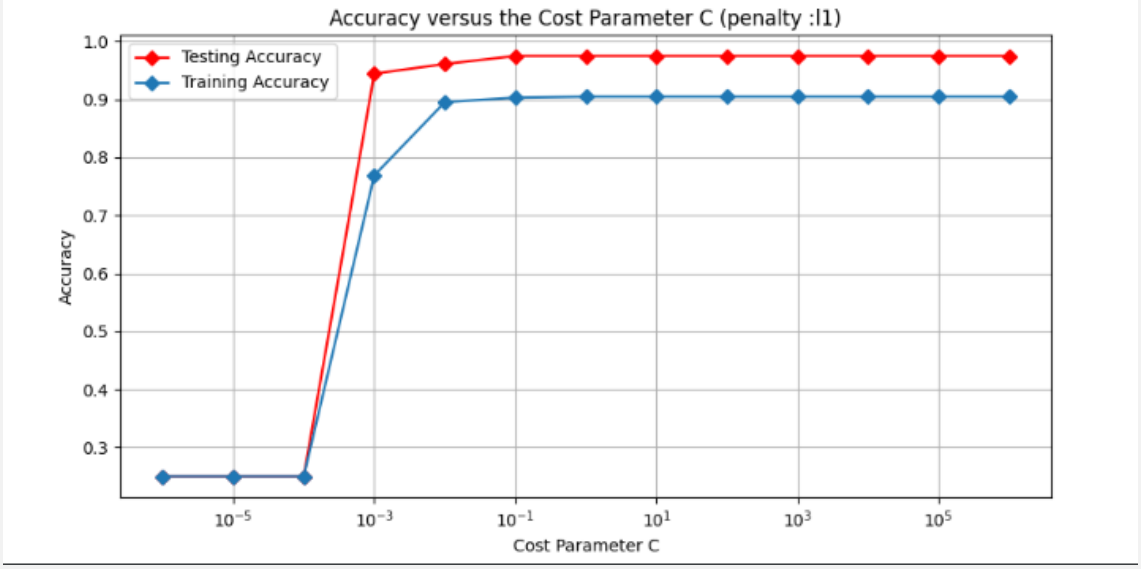


Figure 15: Parameter tuning of penalty 'l1' or 'l2'

4.2 Kernal method

When applying kernel method of rbf and polynomial. We also have penalty parameter c of the error term. If C is too large or too small, the generalization ability becomes worse. The higher C is, the less error is tolerated and the easier it is to overfit. The smaller C is, the less fit it is.

	c	gamma	train_acc	test_acc
0	0.01	0.01	0.887543	0.963889
1	0.01	0.10	0.924740	0.963889
2	0.01	1.00	0.912630	0.358333
3	0.01	10.00	1.000000	0.252778
4	0.01	100.00	1.000000	0.655556
5	0.10	0.01	0.918685	0.975000
6	0.10	0.10	0.949827	0.980556
7	0.10	1.00	0.912630	0.358333
8	0.10	10.00	1.000000	0.252778
9	0.10	100.00	1.000000	0.655556
10	1.00	0.01	0.936851	0.977778
11	1.00	0.10	0.985294	0.988889
12	1.00	1.00	1.000000	0.761111
13	1.00	10.00	1.000000	0.258333
14	1.00	100.00	1.000000	0.250000
15	10.00	0.01	0.963668	0.983333
16	10.00	0.10	0.999135	0.961111
17	10.00	1.00	1.000000	0.788889
18	10.00	10.00	1.000000	0.261111
19	10.00	100.00	1.000000	0.250000
20	100.00	0.01	0.989619	0.991667
21	100.00	0.10	1.000000	0.961111
22	100.00	1.00	1.000000	0.788889
23	100.00	10.00	1.000000	0.261111
24	100.00	100.00	1.000000	0.250000

Figure 16: Kernel_rbf4567

	c	degree	train_acc	test_acc
0	0.01	2.0	0.669550	0.836111
1	0.01	3.0	0.630623	0.822222
2	0.01	4.0	0.539792	0.705556
3	0.01	5.0	0.506055	0.641667
4	0.01	6.0	0.478374	0.588889
5	0.10	2.0	0.889273	0.961111
6	0.10	3.0	0.911765	0.983333
7	0.10	4.0	0.805363	0.930556
8	0.10	5.0	0.782007	0.925000
9	0.10	6.0	0.705017	0.850000
10	1.00	2.0	0.942042	0.975000
11	1.00	3.0	0.961073	0.986111
12	1.00	4.0	0.948962	0.950000
13	1.00	5.0	0.945502	0.963889
14	1.00	6.0	0.909170	0.861111
15	10.00	2.0	0.965398	0.969444
16	10.00	3.0	0.991349	0.977778
17	10.00	4.0	0.990484	0.941667
18	10.00	5.0	0.989619	0.963889
19	10.00	6.0	0.977509	0.900000
20	100.00	2.0	0.978374	0.938889
21	100.00	3.0	1.000000	0.961111
22	100.00	4.0	1.000000	0.941667
23	100.00	5.0	1.000000	0.930556
24	100.00	6.0	0.998270	0.897222

Figure 17: Kernel_poly4567

The other parameter needs to be selected is gamma which can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. It implicitly determines the distribution of data after mapping to the new feature space. The number of support vectors affects the speed of training and prediction. The physical meaning of Gamma is about the width of RBF, which will affect the scope of action of the Gaussian corresponding to each support vector, thus affecting the generalization performance. If gamma is too large, then the radiation range of the support vector is very small, so small that only a small number of samples are affected. At this point, no matter what the value of C is, overfitting cannot be avoided.

If gamma is too small, the model is too limited, and the radiation range of the selected support vector is very large, any one support vector will affect the whole data set, making the model unable to capture more complex patterns. The resulting model is similar to a linear model with a set of hyperplanes that separate the high-density centers of any pair of two classes. We loop c from 10^{-2} to 10^{2} at the same time and the result is shown in figure(16) and figure(17).

We can see from the test results that a smoother model (with a smaller gamma) can be complicated by increasing the C value to fit more samples. Note that for some moderate gamma values, when C becomes very large, we can get the same model as when C is small. However, when the radiation range gamma is appropriate, the kernel can be considered as a good structural regular method. Therefore, on the premise that the performance of the model is good when C is relatively small, it is not inclined to make C very large, because a small C value can save memory and run the program faster. Considerd both the accuracy of the training and testing set, we choose C = 1, gamma = 0.01 for kernel RBF and C = 1, gamma = 3 for kernel polynomial.

4.3 KNN

In knn algorithm we have two parameters. Parameter k for k nearest neighbors, parameter method choosing from "distance" and "uniform" which gives the weight for each distance.
We test these parameters through looping k from 1 to 41, and the two method, evaluating through accuracy.

```
for method in ["uniform", "distance"]:
    for k in range(1, 41):
        knn = KNeighborsClassifier(n_neighbors=k, weights=method)
        knn.fit(trainimgs35, trainlabel35)
        acc = knn.score(testimgs35, testlabels35)
        a1[n, k-1] = acc
        if acc > accmax:
            accmax = acc
            kmax = k
            best_method = method
            test_predict0 = knn.predict(testimgs35)
    n = n + 1
```

Figure 18: code of knn parameters tuning

Figure () shows the accuracy of different k. It is easy to find that k rise at the beginning and decrease after the peak. When k is small, it is easy to fit the local message so the model is easy to get over fit. When k is very large, assume it is N, the model will consider all sample. That is similar to make no predict and is meaningless.

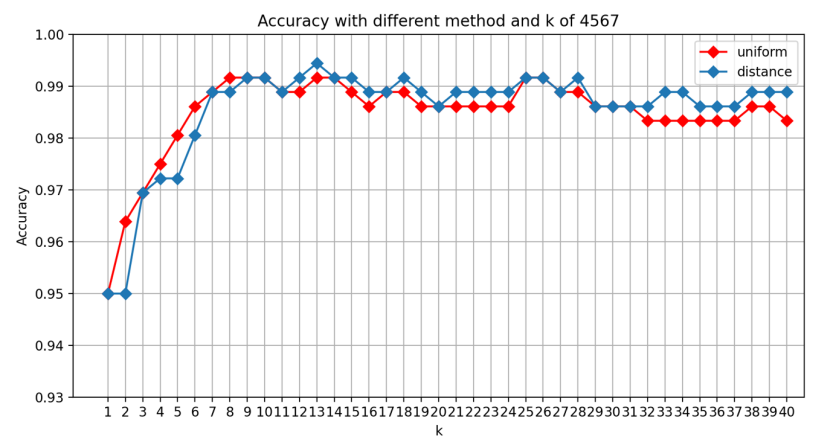


Figure 19: Parameter tuning result of knn

In this figure we can also find the differences between uniform method and distance method from the blue and red line. Obviously, the blue line is sometimes higher than the other one, especially when k grows. As a result, distance method works and makes the accuracy much higher.
For the training set, it is best to choose k = 13 and distance method.

4.4 K-means

In this algorithm we have three parameters, cluster parameter k, random matrix and iteration number i. We already know that the real classes is four, but the best k may not be four. Firstly, it must not be smaller than four. This is because if it is smaller than four, the samples belong to different classes will be mixed together into one cluster. The reason for bigger k is better is, some personalized number may be divided to different clusters. For example, someone write the number really fat, but someone will write it thinner. With clusters more than four, it can be divided to more specific classes. When classifying, it can identify more detailed features, and the accuracy may be increased. So we can test K from 4 to 25. On the one hand, the best answer can be found in the vast majority of cases, on the other hand the computing power of the laptop has reached the upper limit and takes a long time. We can see in the figure that K increases first and then decreases. It is obvious that we take K = 19 as the most appropriate.

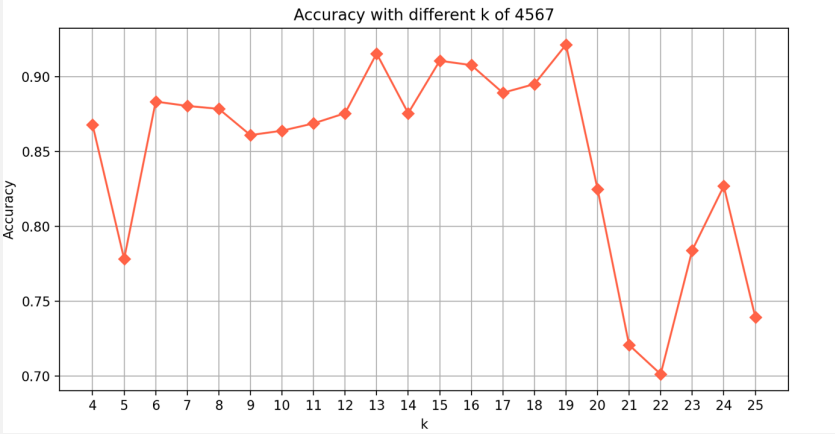


Figure 20: Parameter tuning result of k-means

Random matrix is also important because the random matrix is used to determine the center point

at the beginning of the algorithm, in the later application, if the random matrix is different, the results will fluctuate greatly.After determining the value of K, we try to choose a random matrix with higher accuracy and keep it for later testing. We took 15 random matrices in the code and calculated the accuracy rate, keeping the better values. Fifteen times is the value we determined after testing, and we found that we can find a random matrix within 15 times whose accuracy can stably reach about 90%.As you can see from this diagram of the selection of random matrices, the highest value is 0.93, and the other values is also around 0.9.

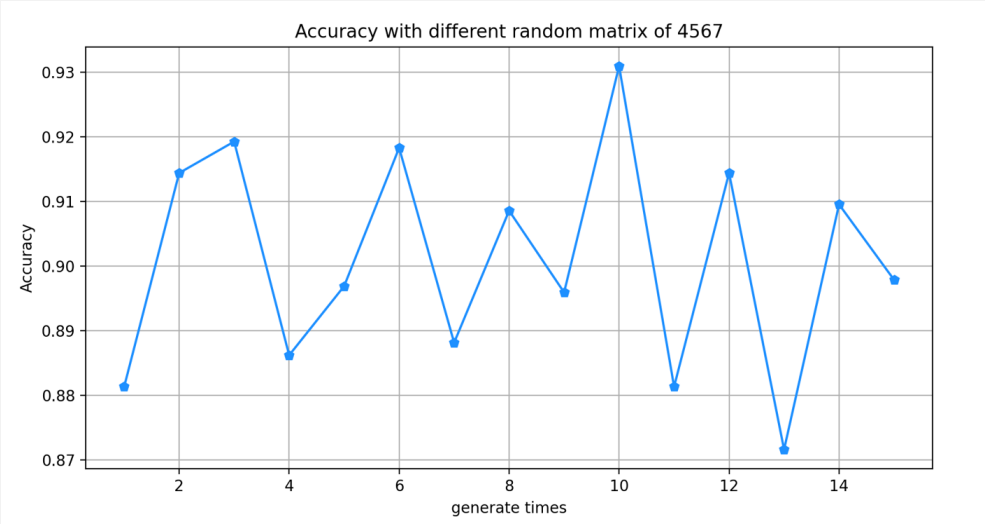


Figure 21: Parameter tuning result of random matrix

Finally, we need to choose the number of iterations. In a package of sklearn, the default number of iterations is 300. However, in the case of a specified K and random matrix, the number of iterations may be smaller or need to be larger. And in this project, we have fewer categories, it is therefore possible to use a smaller value. As you can see from the chart, we don't need a lot of iterations. With the increase of the iterations, the accuracy of the experiment is gradually improved, and tends to be stable at about 13 times.Finally, it reached its peak at 16 times. This value is much smaller than 300, which can greatly reduce the calculation time and improve the calculation efficiency. The latter change is very small, so the number of iterations has a negligible effect on it.

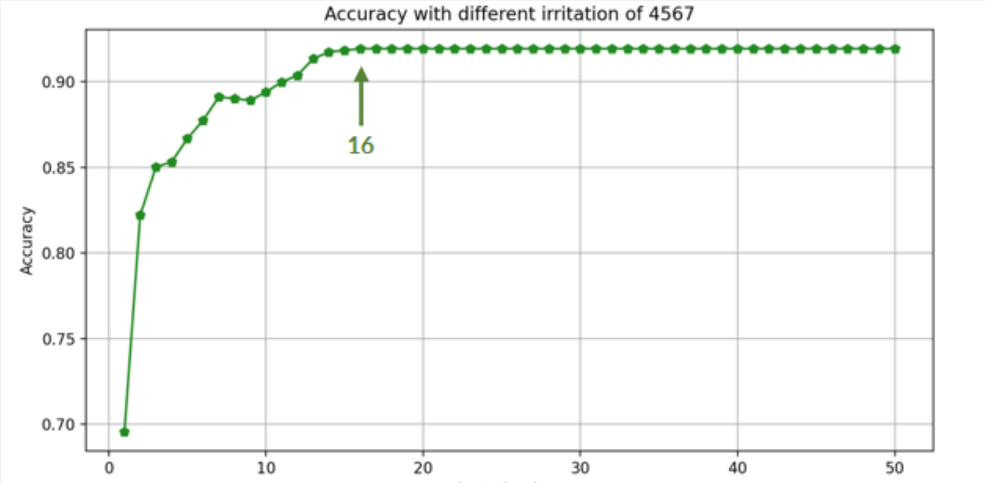


Figure 22: Parameter tuning result of iteration parameter

Here we only take the classification of 4567 as an example. Since we do not store a preliminary center matrix, we will get different values for each experiment and different charts are presented. The trend and principle are similar each time. Therefore, although the exact best value can not be obtained, these methods have improved the accuracy to a certain extent.

4.5 Random forest algorithm

There are many parameters that can be adjusted in random forest classifier. This project improved the accuracy rate of 4567 classification from 92 % to 99 % by adjusting three of the most critical parameters.

(1)n _ estimators describe the number of decision trees in the forest, and by adjusting the values of the parameters, we found that the highest accuracy rate at 151 was 96 % .

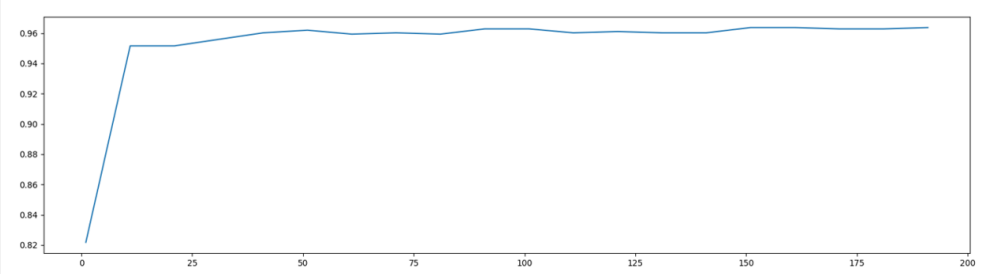


Figure 23: The result of adjusting n _ estimators

(2) max_depth defaults to None, which is the maximum depth. Setting the depth to limited reduces the accuracy of the model, but reduces the overall running time. The time reduction is not obvious, so select None by default for maximum accuracy.

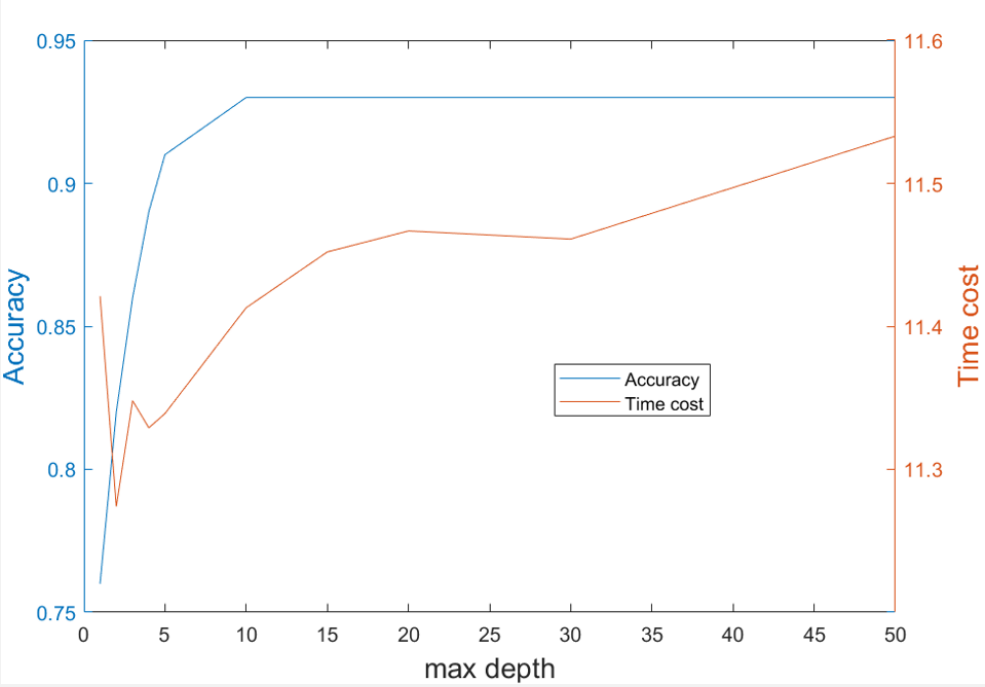


Figure 24: The result of adjusting max_ depth

(3) max_features is used to describe the complexity of the model. Combined with samples, the maximum accuracy rate is 98 % when the parameter is set to 1 .

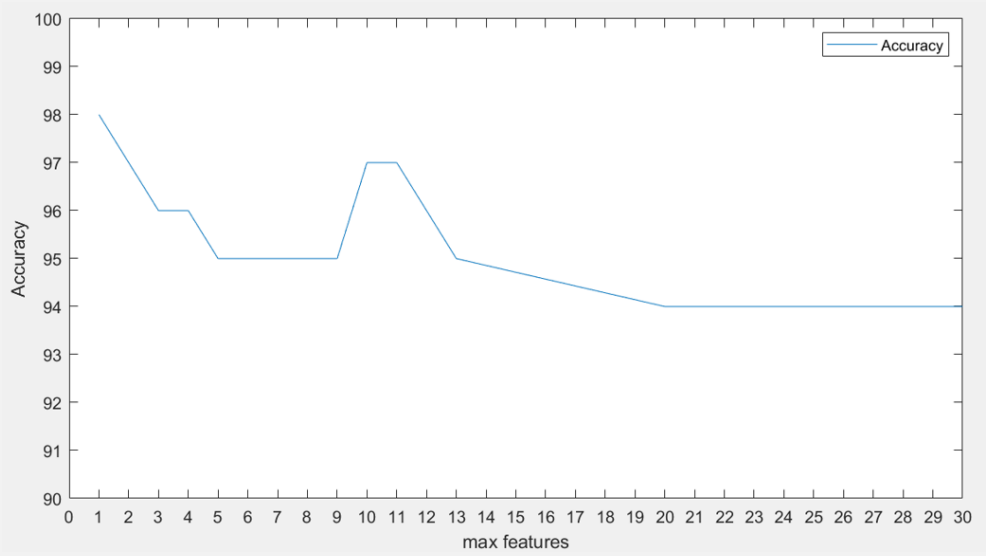


Figure 25: The result of adjusting max _ features

The code after adjusting the parameters:

```
rfc = RandomForestClassifier(n_estimators=151, n_jobs=-1, random_state=90, max_depth=None, max_features=1)
rfc.fit(x_train, y_train)
pre = rfc.predict(x_test)
report = classification_report(y_test, pre)
```

Figure 26: Adjusted code

Comparison of results:

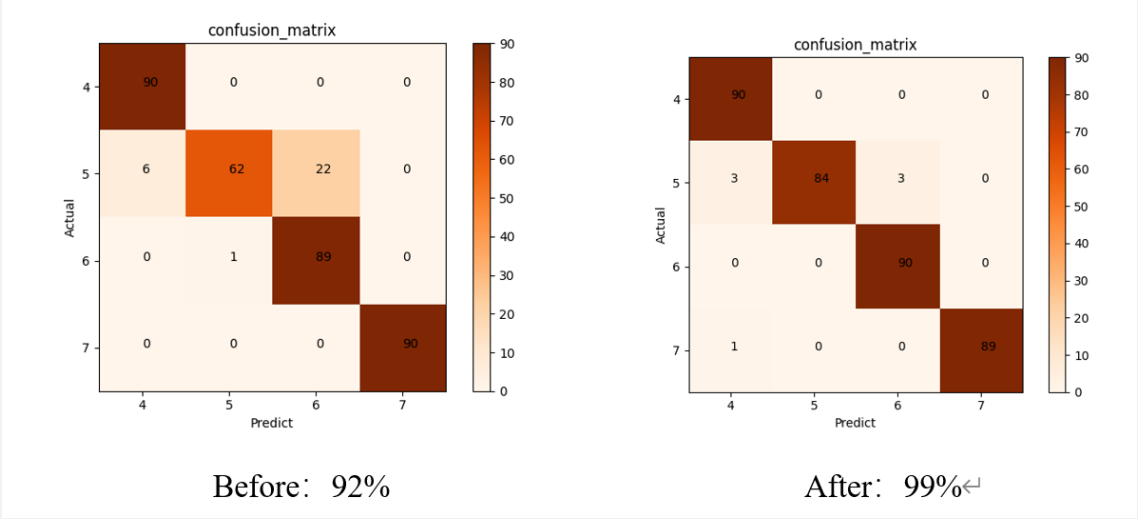
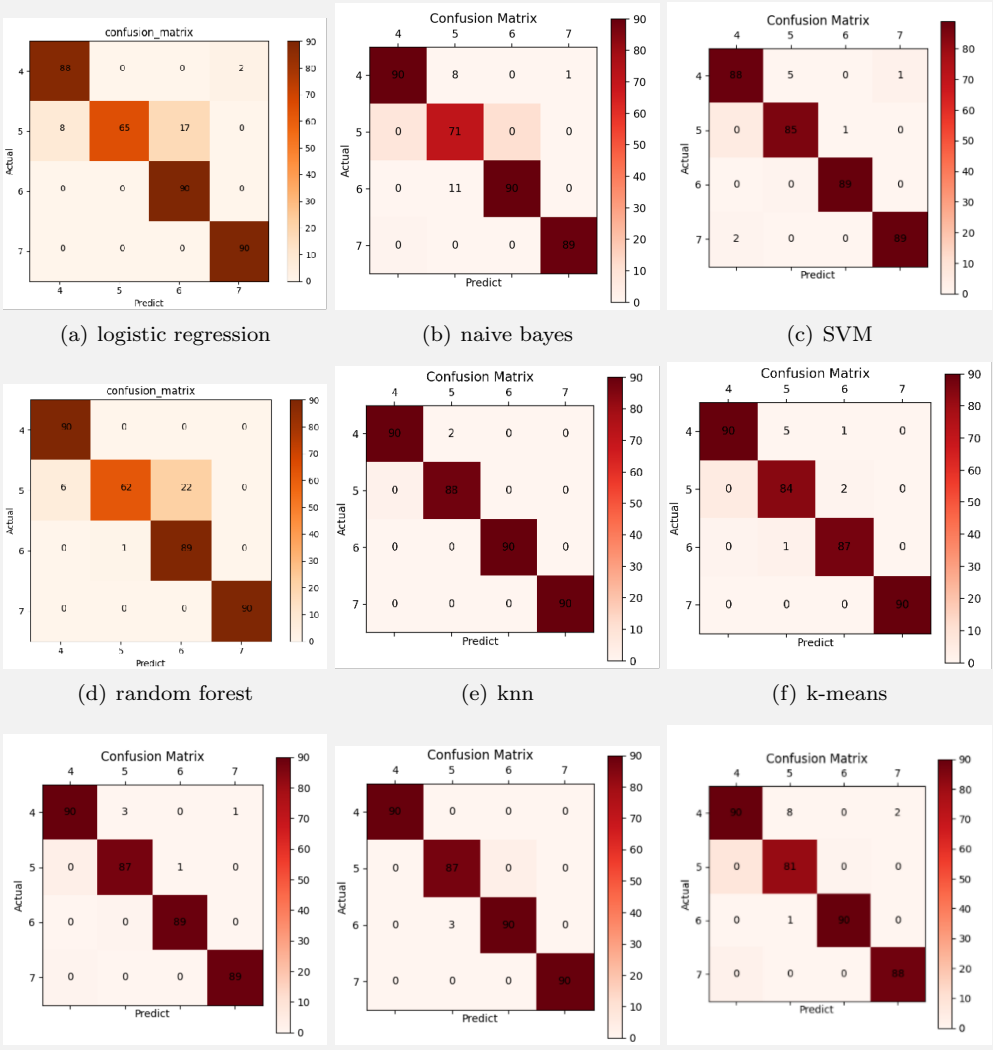


Figure 27: Comparison of accuracy before and after adjusting parameters

5 Result and analysis

5.1 Visualised result

In order to show the classification accuracy and error more intuitive, we draw confusion matrixes for the classifications. Take classification of 4567 as an example we have:



(a) logistic regression (b) naive bayes (c) SVM

(d) random forest (e) knn (f) k-means

(g) Kernel poly (h) Kernel rbf (i) MLP

Figure 28: Confusion matrixes of classify 4567

.

As this project is to carry out binary classification of 35 and 89, multiple classification of 4567. We have summarized the accuracy and time of all methods and all classifications in the following table. The result of before and after data processing is also concluded.

	Training set accuracy		Testing set accuracy		Time	
Algorithm	before data optimzation	after data optimzation	before data optimzation	after data optimzation	before data optimzation	after data optimzation
log35	49.82%	89.40%	50%	100%	1.618	16.129
log89	49.31%	94%	50%	97.22%	1.596	16.221
log4567	32%	86.30%	25.83%	92.50%	3.182	32.683
fore35	100%	100%	59%	99%	0.164	1.643
fore89	100%	100%	76%	96%	0.161	1.622
fore4567	100%	100%	67%	92%	0.169	1.662
SVM35	100%	92.40%	50%	99.40%	2.141	0.002
SVM89	100%	97.20%	68.33%	96.10%	0.558	0.002
SVM4567	100%	90.48%	52.22%	97.50%	16.078	0.004
Kernel_rbf35	100%	98.61%	51.67%	99.44%	0.173	0.037
Kernel_rbf89	100%	100%	51.10%	99.44%	0.189	0.022
Kernel_rbf4567	100%	98.96%	26.67%	99.17%	0.823	0.04
Kernel_poly35	100%	100%	53.89%	100%	0.063	0.031
Kernel_poly89	100%	96.10%	67.22%	98.33%	0.049	0.019
Kernel_poly4567	100%	96.11%	51.94%	98.61%	0.313	0.027
naive bayes35	60.58%	90.38%	42.22%	99.44%	3.399	3.396
naive bayes89	67.41%	93.97%	68.89%	92.78%	3.411	3.5
naive bayes4567	53.01%	89.98%	48.89%	94.44%	7.696	7.659
knn35	100%	100%	72.22%	98.89%	1.354	0.921
knn89	100%	100%	81.11%	99.44%	1.372	0.968
knn4567	100%	100%	75.83%	99.44%	3.385	2.629
kmeans35	66.90%	90.57%	66.11%	98.33%	14.949	4.751
kmeans89	71.18%	95.29%	65.56%	98.33%	11.511	5.029
kmeans4567	55.25%	92.60%	54.16%	97.78%	60.513	16.499
MLP35	50.52%	95.32%	49.44%	98.33%	0.538	0.746
MLP89	57.39%	97.79%	52.78%	96.67%	0.503	0.755
MLP4567	25%	96.54%	25%	96.94%	0.947	1.534

Figure 29: Result table of work

To make the difference more obvious, we also made them to column diagram.

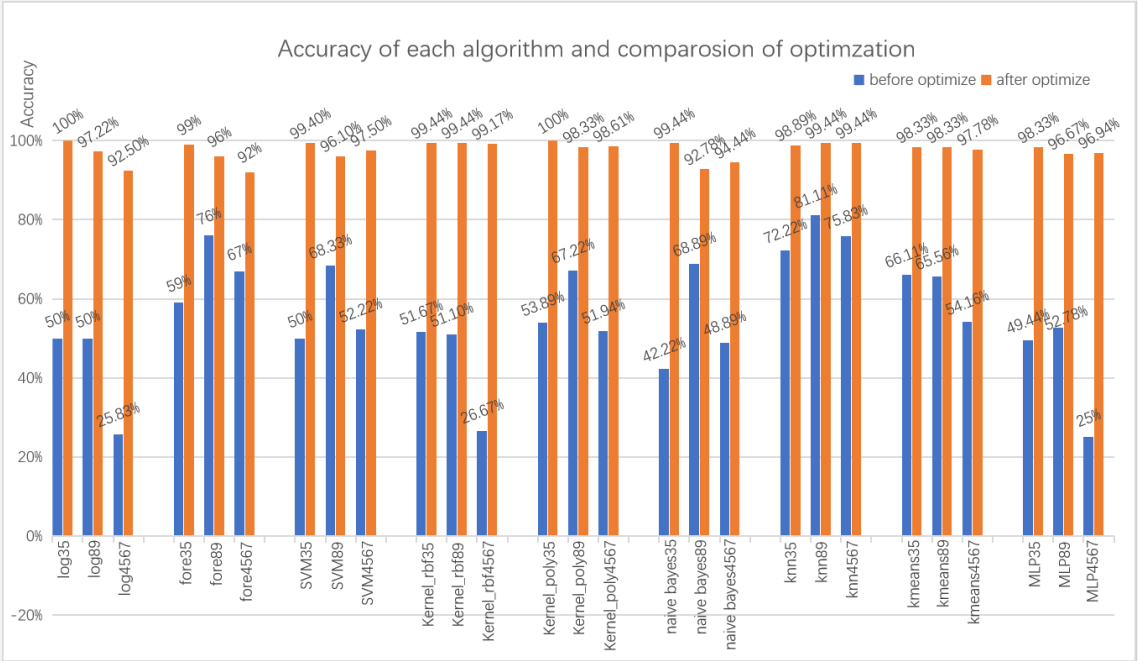


Figure 30: Result in form of column diagram

To restate, all the data and result mentioned in this report are related to the testing set upload in blackboard. From those figures, it is obvious that we make great progress on increasing the accuracy. The data processing methods really make sense. Moreover, we are delighted to summarize that all accuracy are higher than 92%, and the highest accuracy in our project reached 100%! Kernel method

seem to reach the best general effect.

5.2 Reason of doing optimization

1.The reason for optimize image informationBy making the edge of the image clearer through binarization and other methodsit can improve the accuracy of image recognition. Images can be moved and repositioned by stretch and cutting so that features of the same class are represented in similar pixels and therefor they can be better recognized by machines.
2.The reason for data dimensional reduction: Data dimensions and storage space required are reduced. Methods like PCA and LDA can eliminate redundant variables, improve the accuracy of the algorithm and save training time of the model. Result visualization is also more convenient after dimensional reduction.
3.The reason for cross-validation:It can help to evaluate the predictive performance of models, especially the performance of trained models on testing set, which can reduce overfitting and lift stability. It can also select parameters from limited data and extract as much valid information as possible.
4.The reason for adjust parameters: Parameter adjustment can help us improve accuracy and further understand the characteristics of the model.

5.3 Discussion of the two image optimization

There are many similarities and differences between the two image processing methods. From aim of image processing, both of them are to unify the bizarre position and the different size of the numbers in the image and get a result of binarized image. In terms of computational steps, the L algorithm has fewer steps than the S algorithm. But in terms of accuracy and effect, the S algorithm is higher than the L algorithm.Generally speaking, the accuracy of S algorithm is about 8% higher than that of L algorithm. We compared the accuracy of the two algorithms when using Naive Bayes to classify 35, and the calculated results are as follows:

```
before optimize:The accruacy socre of 35 is  0.6057692307692307
The accruacy test socre of 35 is  0.4222222222222222
LIU alogorithm:The accruacy socre of 35 is  0.8153846153846154
The accruacy test socre of 35 is  0.9333333333333333
SU algorithm:The accruacy socre of 35 is  0.9038461538461539
The accruacy test socre of 35 is  0.9944444444444445
```

Figure 31: Compare of image algorithms

Both of them have been greatly improved compared with no image processing ones, and we notice that the S algorithm finally makes the accuracy of the test results as high as 99%.Therefore, S algorithm is used in all of our algorithms for image optimization.
When compared with the L algorithm and all the algorithms in other groups we heard in the meeting, we get the reason why the S algorithm can make the accuracy higher. Compared with other algorithms, most of them only complete denoising and graph centralization at the beginning, but dose not binarize the image after processing. As a result, the numerical boundaries of their graphs are still blurred, which leads to some errors in the calculation of each algorithm.Compared with the L algorithm, the S algorithm have one more step of streching stretching. The original purpose of the L algorithm is to preserve the diversity of digital writing. But from the test results, the machine's learning effect on diversity is not as good as expected.Therefore, in fact, the stretching step of the S algorithm is to remove the diversity and personalization of the numbers. For example, all the flat circles of numbers in his house are drawn to a similar size and curvature. This operation is more conducive to classification.

6 Completion experience

1. In the whole project, we first learned the mathematical principles of Logistic regression, Naive Bayes Classifier and SVM with Different Kernels. After that, Logistic and Bayes classifiers were implemented by handwritten codes, and SVM classifiers were implemented by invoking mathematical packages. After completing the three basic classification algorithms, we continued to learn more advanced machine learning classification algorithms, such as Random Forest, KNN, K-means and MLP, and had a preliminary understanding of their mathematical principles and logical system. After that, the task of classifying handwritten numbers is realized by using a mixture of calling code package and handwritten code for each algorithm. In the whole process, we not only reviewed what we learned in class and applied it to practical problems, but also exercised our ability to write python code and learn independently . We benefited a lot.

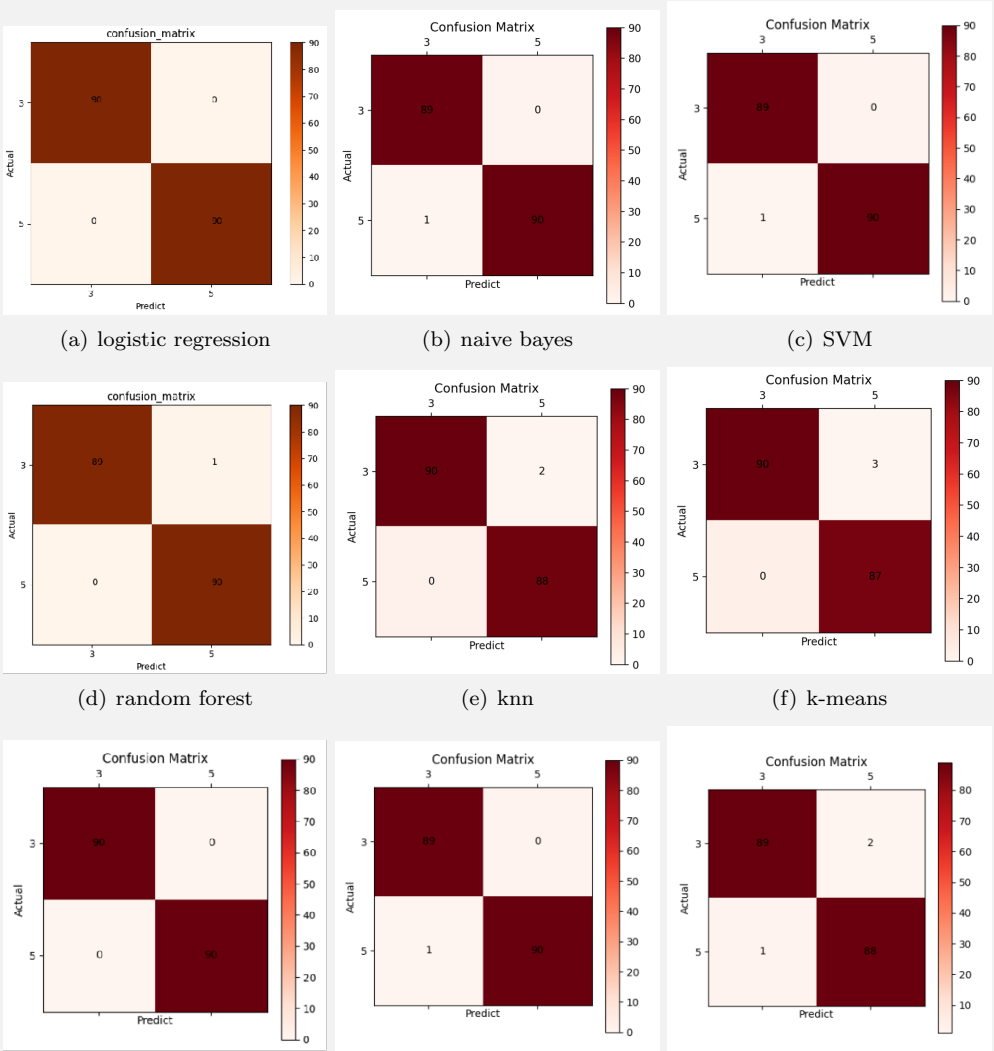
2. During the whole process of completing the project, we all agree that the pre-processing of data is crucial. When using the original data set provided by the faculty team, the test accuracy of our classification algorithm was very low, roughly between 20 % and 50 % . At this time, our team came up with the idea of image processing, and obtained new data sets by means of binarization, PCA and other processing operations. After that, we use the optimized data set for learning and training, the effect is remarkable, and the accuracy rate is increased to an amazing 90 %. The cross-validation method improves the stability of the results and reduces the variance.

3. We also discussed how to further improve the operation efficiency and classification accuracy by adjusting parameters. Taking the Random Forest algorithm as an example, the accuracy rate was only 92% when the default value was adopted for multi-classification. After that, we learned the parameter representation, meaning and usage in detail, and adjusted the parameters patiently, so that the final accuracy rate was improved to 99 %, the effect of adjusting parameters was remarkable.

4. Finally, we visualized the results of classification and drew the confusion matrix, which clearly reflected the classification effect and characteristics of each algorithm. In addition, the training and testing speed and accuracy of various algorithms were compared horizontally, the kernel method was finally concluded as the algorithm with the best performance.

7 Appendix

Here are the confusion matrix image for classification of 3 and 5.



(a) logistic regression (b) naive bayes (c) SVM

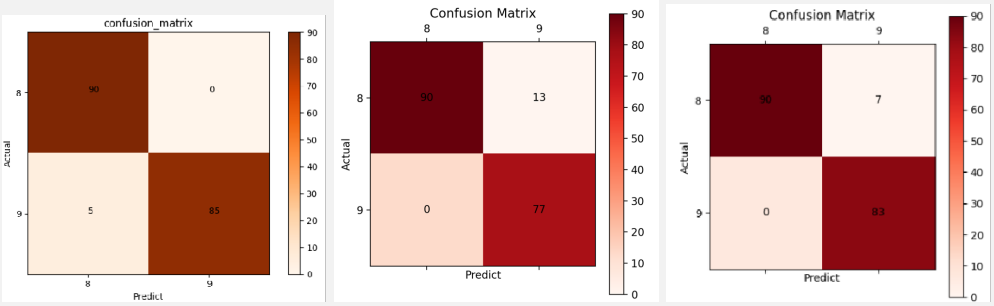
(d) random forest (e) knn (f) k-means

(g) Kernel poly (h) Kernel rbf (i) MLP

Figure 32: Confusion matrixes of classify 3 5

.

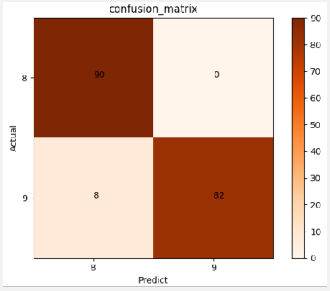
Here are the confusion matrix image for classification of 8 and 9.



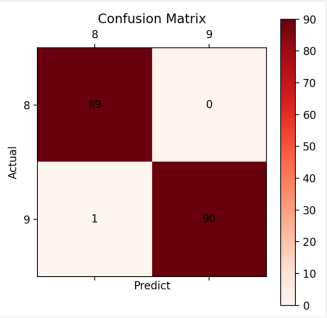
(a) logistic regression

(b) naive bayes

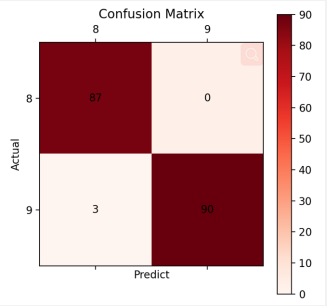
(c) SVM



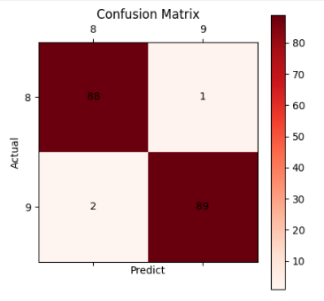
(d) random forest



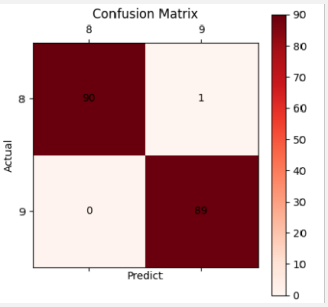
(e) knn



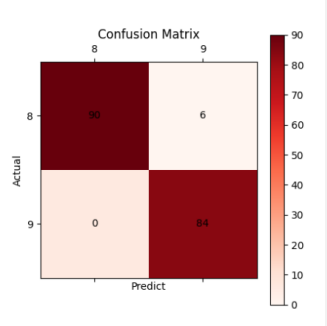
(f) k-means



(g) Kernel poly



(h) Kernel rbf



(i) MLP

Figure 33: Confusion matrixes of classify 8 9

.