Machine Learning-32513-Assignment 2

Fan Feng 12832022

XinChen Wang 12772149

Content

Introduction

This report will use machine learning algorithms to explore the classic handwritten digit recognition data set (mnist datasets), a handwritten digital image database, each of which is a single number from 0-9.
We performed experiments on the mnist dataset using algorithms such as K-Nearest Neighbor (KNN) algorithm, Logistic Regression (LR) algorithm, and Support Vector Machine (SVM). Our code running environment is google colab's Jupyter Notebook, the running language is python, mainly using the neighbors, metrics and linear_model of the sklearn module. The code source is added in GitHub, the link is:
https://github.com/shuyeff/UTS_ML2019_ID12832022
The accuracy of the experimental results is satisfactory, and the running time is moderate. In addition, we ran the same code in the PyCharm environment with the same accuracy and slightly improved running time. The experimental results show that the accuracy of KNN algorithm is 96.25%, the accuracy of LR algorithm is 91.69%, and the accuracy of SVM algorithm is 98.57%.
It is very meaningful to study the mnist dataset, because handwritten digit recognition can be extended to letter recognition and language recognition. It has widely applications in many practical application scenarios, such as illegal vehicle license plate recognition and handwritten language machine translation.

Exploration and Methodology

The challenge of the machine learning model consists of two main points: overfitting and underfitting.
For under-fitting, the representative model does not have a good fit for the dataset, so the possible optimization points are: using more complex models, complex models require more parameters to learn, and often can fit more complex data. set. It may also be related to sample and feature engineering, such as enriching samples and providing better feature engineering for algorithms.
For over-fitting, the representative model over-fits the dataset. The

corresponding methods have simplified model parameters, increases regularization terms, and reduces noise in the dataset. When using deep learning techniques, you can also use dropout techniques to avoid overfitting.

Input

The input of this task is a handwritten digit recognition data set (mnist datasets), which contains a total of 70,000 handwritten digital grayscale pictures, 60,000 sheets as a training set, and 10,000 sheets as a test set, where the numbers include 0 to 9, from the perspective of the picture, shown in Figure 1.

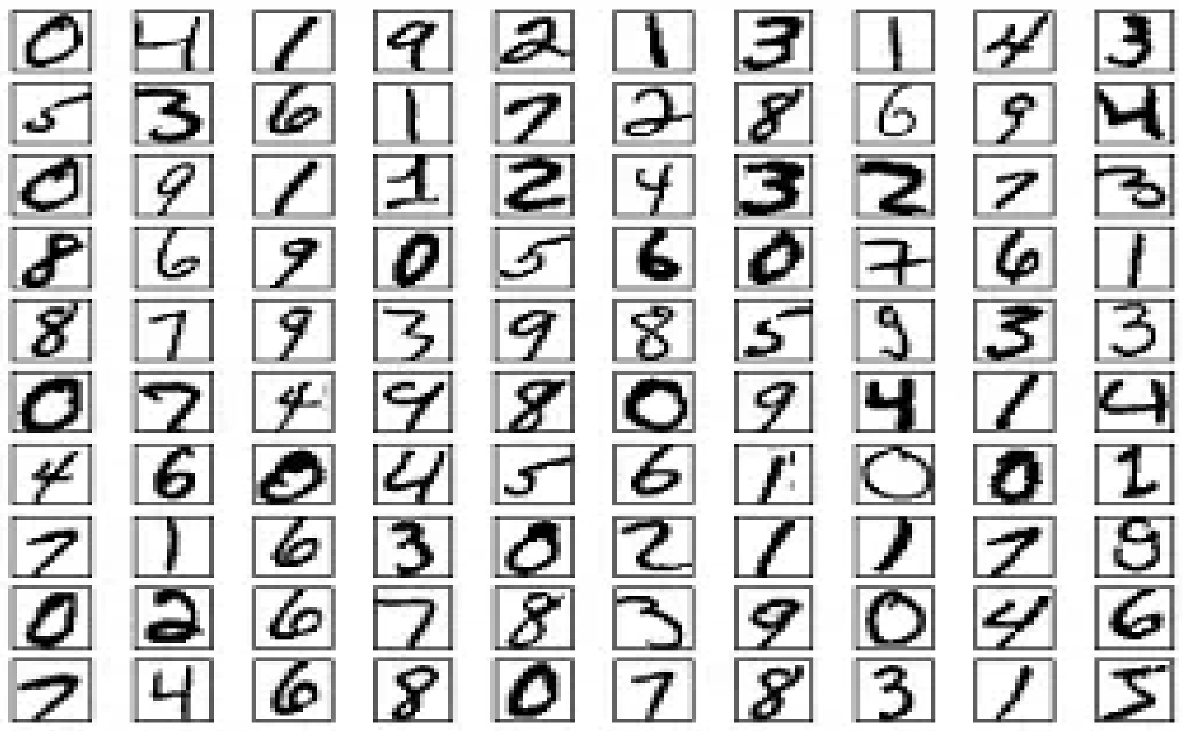


Figure 1 part of mnist data setscreenshot

Since each person's writing of the same number is likely to be different, our task is not only to identify the difference between different numbers, but also to identify the difference between the different numbers of the same number, as shown in Figure 2, the same number 7 Different people's writing methods are also different.

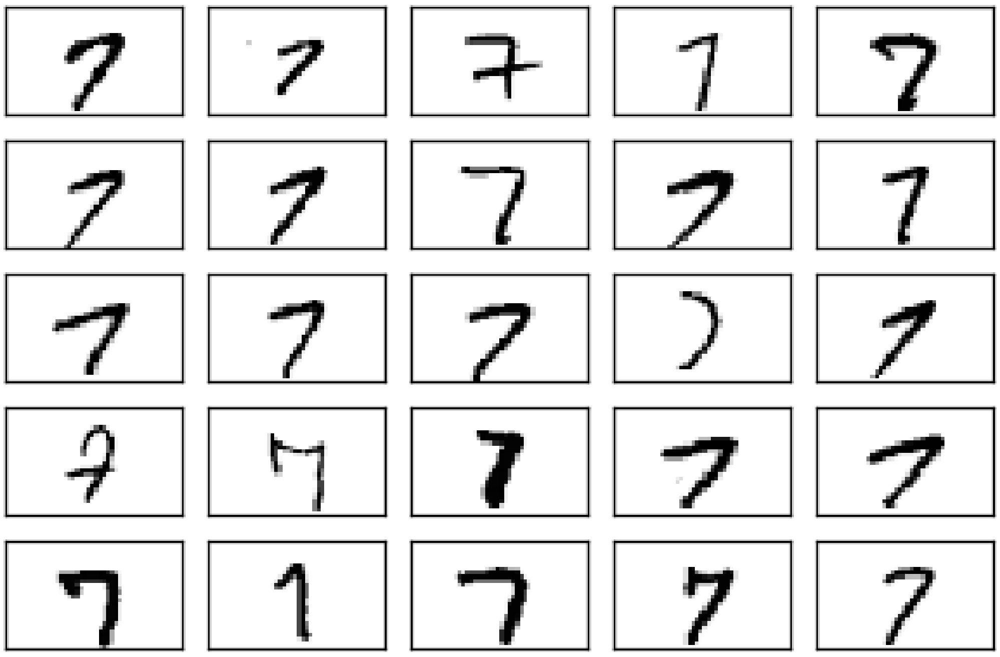
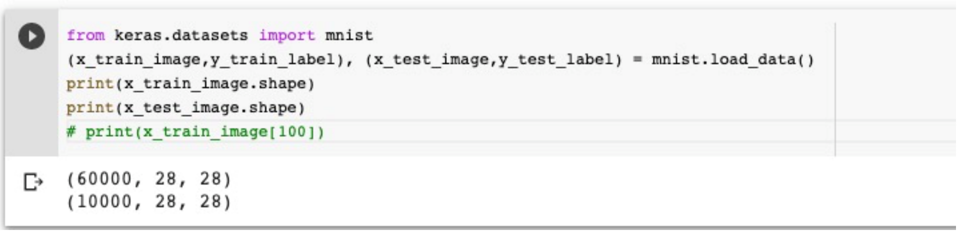


Figure 2 Different ways of writing the number "7"

The format of mnist is grayscale image, so each picture in the computer is stored as a 28*28 matrix. The values in the matrix represent the gray value of each pixel, 255 for black, 0 for white, others can be seen as gray. Download the dataset and run the code on the colab platform, you can see the whole picture of the data set more clearly, as shown in Figure 3, the shape of the training set is (60000, 28, 28), representing a total of 60,000 pictures in the training set, each The picture is composed of 28*28 grayscale pixels, and the same test set is composed of 10,000 pictures.

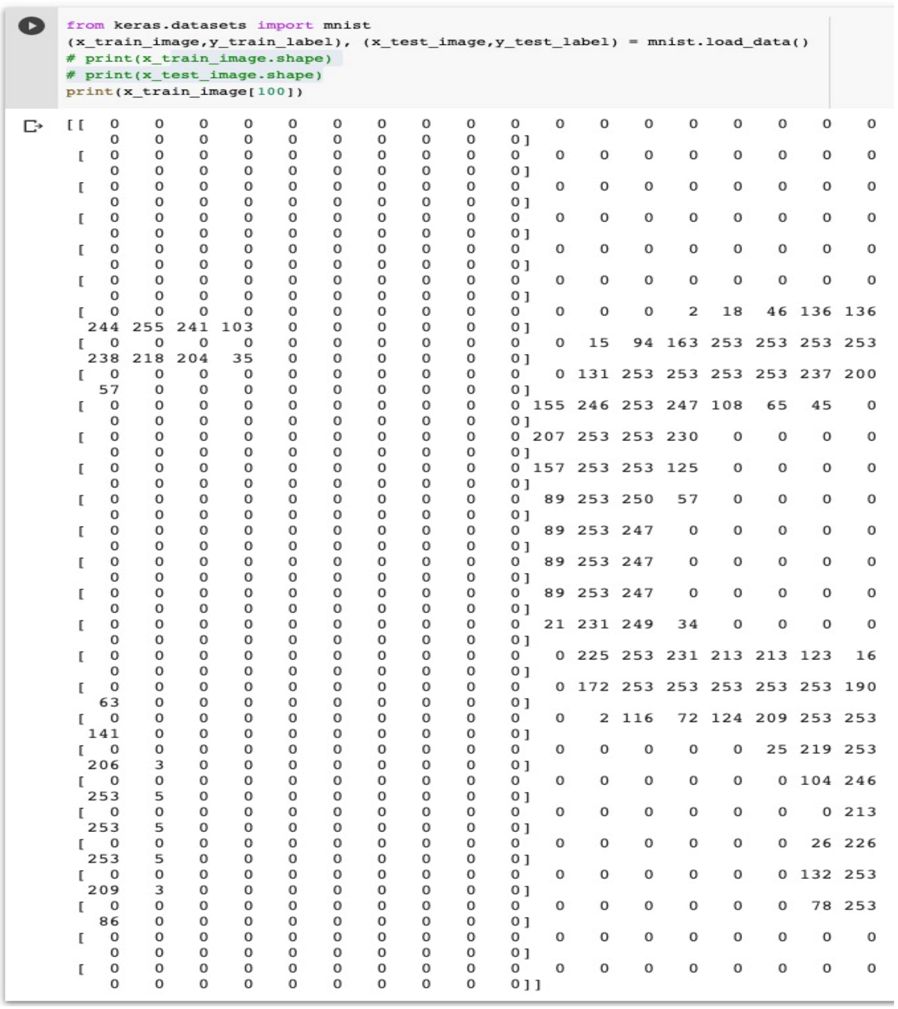


```
from keras.datasets import mnist
(x_train_image,y_train_label), (x_test_image,y_test_label) = mnist.load_data()
print(x_train_image.shape)
print(x_test_image.shape)
# print(x_train_image[100])
```

```
(60000, 28, 28)
(10000, 28, 28)
```

Figure 3 data set shape

We randomly output one of the pictures, as shown in Figure 4.



```python
from keras.datasets import mnist
(x_train_image,y_train_label), (x_test_image,y_test_label) = mnist.load_data()
# print(x_train_image.shape)
# print(x_test_image.shape)
print(x_train_image[100])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   2  18  46 136 136
  244 255 241 103   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  15  94 163 253 253 253 253
  238 218 204  35   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 131 253 253 253 253 237 200
   57   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 155 246 253 247 108  65  45   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 207 253 253 230   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 157 253 253 125   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  89 253 250  57   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  89 253 247   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  89 253 247   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  89 253 247   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  21 231 249  34   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 225 253 231 213 213 123  16
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 172 253 253 253 253 253 190
   63   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   2 116  72 124 209 253 253
  141   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  25 219 253
  206   3   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 104 246
  253   5   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 213
  253   5   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  26 226
  253   5   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 132 253
  209   3   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  78 253
   86   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
```

Figure 4 one of the grayscale image in mnist

Output

Our task is to train the model to fit the data set, and then calculate a number for a new 28*28-dimensional handwritten digital image. Since machine learning should be trained on the training set, test on the test set. The test set contains 10,000 images, so our output should be the number represented by the 10,000 images. The format should be a one-dimensional vector of length 10000.

K-Nearest Neighbor (KNN)

Algorithm Introduction

kNN, k-NearestNeighbor algorithm. It is widely used in the data mining classification field and is an efficient and simple algorithm. The idea of implementation is relatively simple. K nearest neighbors, no doubt, the value of K is definitely crucial. The principle of KNN is to determine which category x belongs to when predicting a new value x based on what category it is from the nearest K points. KNN is a non-parametric, inert algorithmic model. The meaning of non-parameters does not mean that the algorithm does not require parameters, but rather that the model does not make any assumptions about the data, as opposed to linear regression (we always assume linear regression is a straight line). That is to say, the model structure established by KNN is determined according to the data, which is more in line with the reality. After all, the situation in reality is often inconsistent with the theoretical assumptions. Logistic regression requires a lot of tranning of the data before finally getting an algorithm model. The KNN algorithm does not need it. It does not have a clear process of training the data, or the process is very fast. Simple and easy to use, KNN is a relatively straightforward algorithm compared to other algorithms. Even without a high mathematical foundation, you can figure out its principles. The model training time is fast and the prediction effect is good. Not sensitive to outliers. But the memory requirements are higher because the algorithm stores all the training data. The green dot in the graph is the one we want to predict, assuming K=3. Then the KNN algorithm will find the three closest points to it. Look at which category is more. For example, in this example, there are more blue triangles, and the new green dots are classified into the blue triangle. As shown in Figure 5.

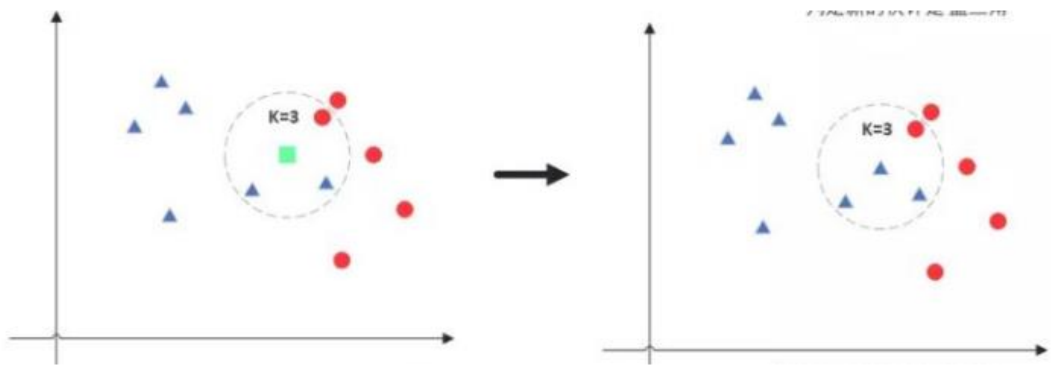
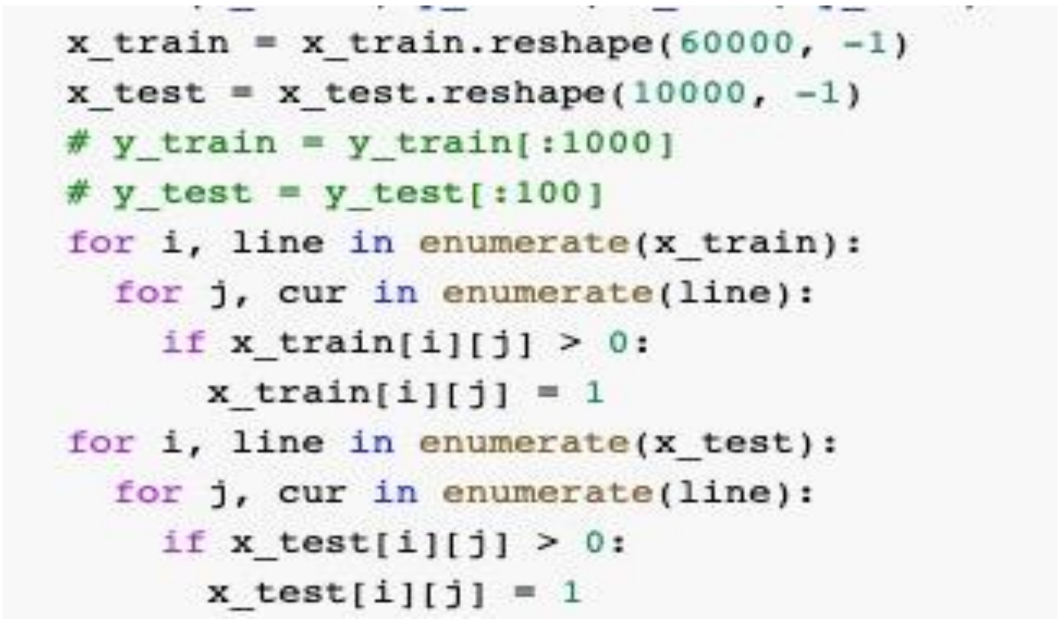


Figure 5

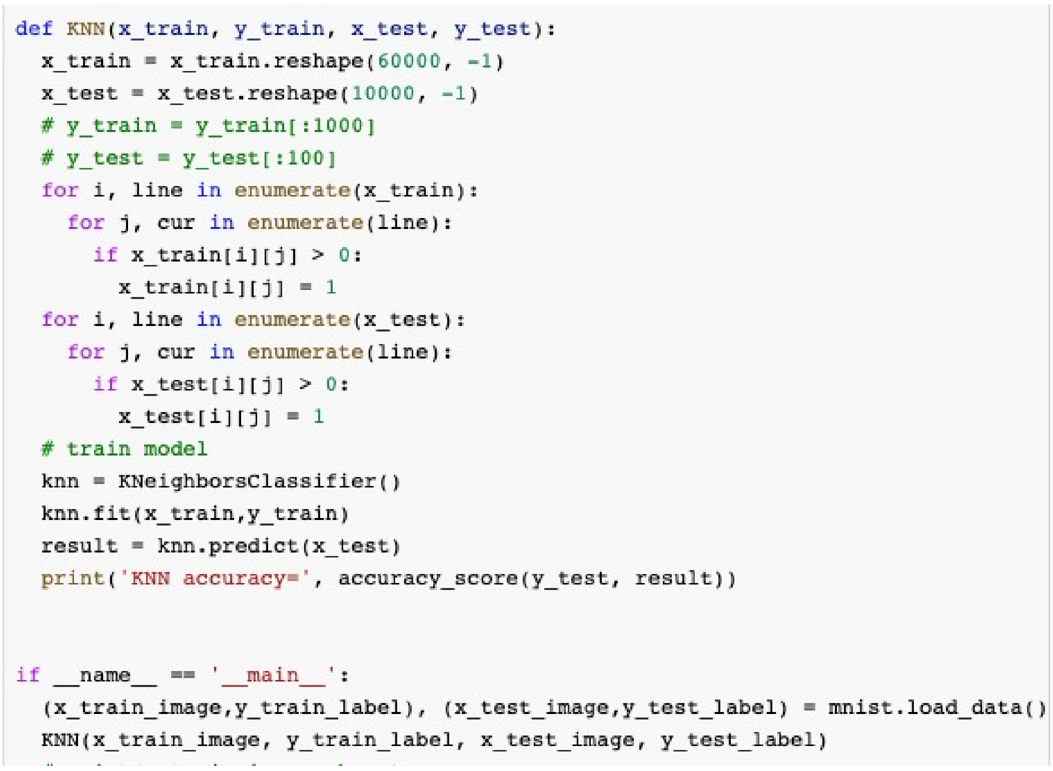
Algorithm Implementation

KNN can be easily applied to the mnist dataset task. We first reshape the image format from (60000, 28, 28) to a two-dimensional matrix of (60000, 28*28), since each image is grayscale picture, then we convert the gray value to 0 or 1, that is, the part with color is 1, and the part without color is 0. The specific code is shown in Figure 6. Then any two pictures can be calculated using the Euclidean distance formula to calculate the distance. The overall KNN code is shown in Figure 7.



```
x_train = x_train.reshape(60000, -1)
x_test = x_test.reshape(10000, -1)
# y_train = y_train[:1000]
# y_test = y_test[:100]
for i, line in enumerate(x_train):
  for j, cur in enumerate(line):
    if x_train[i][j] > 0:
      x_train[i][j] = 1
for i, line in enumerate(x_test):
  for j, cur in enumerate(line):
    if x_test[i][j] > 0:
      x_test[i][j] = 1
```

Figure 6 KNN reshape and normalization code



```
def KNN(x_train, y_train, x_test, y_test):
  x_train = x_train.reshape(60000, -1)
  x_test = x_test.reshape(10000, -1)
  # y_train = y_train[:1000]
  # y_test = y_test[:100]
  for i, line in enumerate(x_train):
    for j, cur in enumerate(line):
      if x_train[i][j] > 0:
        x_train[i][j] = 1
  for i, line in enumerate(x_test):
    for j, cur in enumerate(line):
      if x_test[i][j] > 0:
        x_test[i][j] = 1
  # train model
  knn = KNeighborsClassifier()
  knn.fit(x_train,y_train)
  result = knn.predict(x_test)
  print('KNN accuracy=', accuracy_score(y_test, result))


if __name__ == '__main__':
  (x_train_image,y_train_label), (x_test_image,y_test_label) = mnist.load_data()
  KNN(x_train_image, y_train_label, x_test_image, y_test_label)
```

Figure 7 The overall KNN code

Logistic Regression (LR)

Algorithm Introduction

Although Logistic Regression literally means regression, it is essentially a classification model and is often used for two classifications. Logistic Regression is loved by the industry for its simplicity, parallelism, and interpretability. The essence of Logistic regression is to assume that the data obeys this distribution and then use the maximum likelihood estimation to make an estimate of the parameters. The Logistic distribution is a continuous probability distribution. As shown in Figure 8.

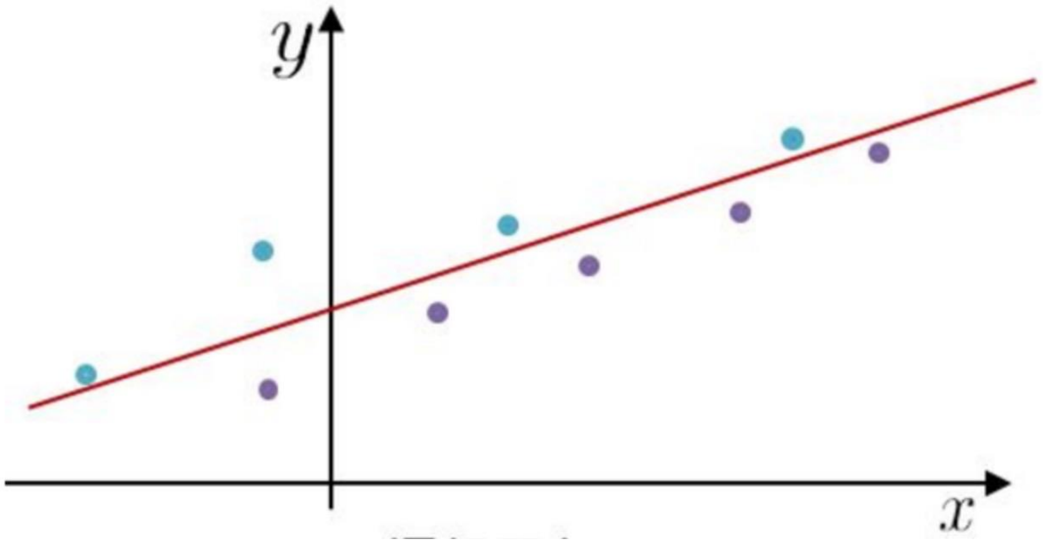


Figure 8 Logistic regression classification line

The Logistic regression expression is:

$$\boldsymbol{h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}}$$

The sigmoid function is shown in Figure 9.

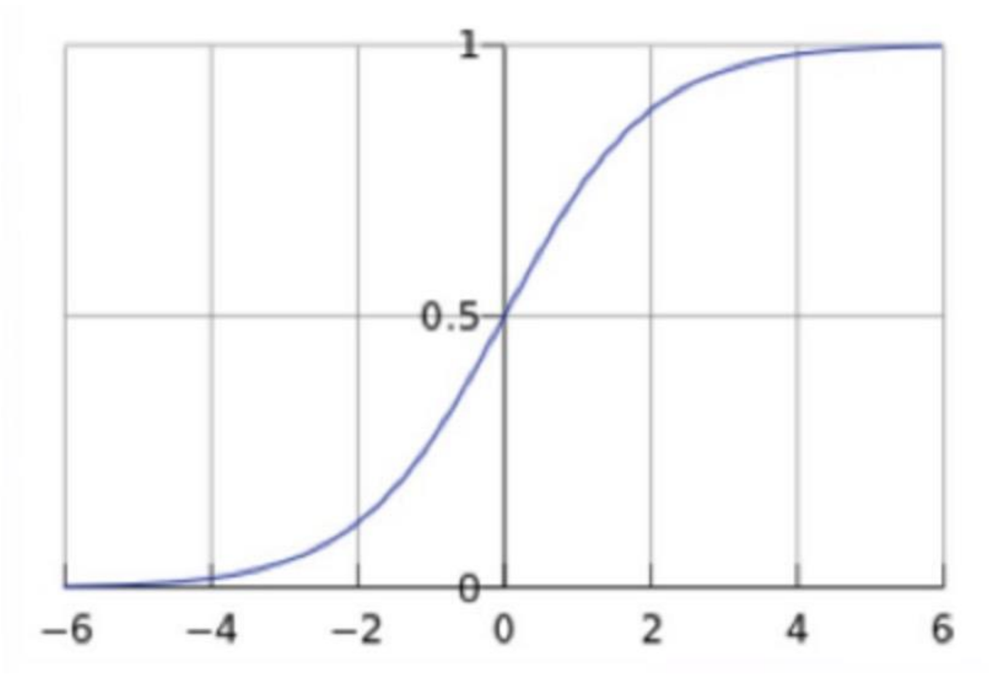
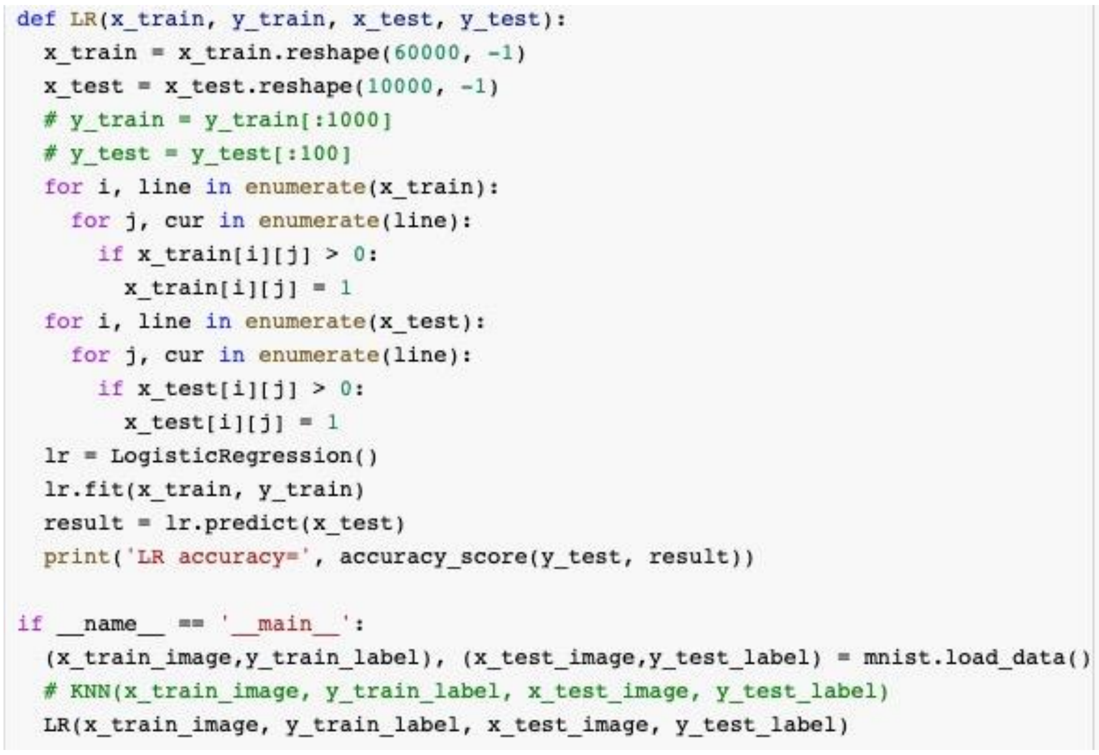


Figure 9 sigmoid function image

Algorithm implementation

Logistic regression is a classification model that can also be applied to the mnist dataset, but usually the general logistic regression can only be applied to the Binary Classification problem. To achieve classification of multiple categories, we must improve the logistic regression. It adapts to multiple classification problems. One way is to create a two-classifier directly for each category, with a sample with this category labeled 1 and a sample with other categories labeled 0. If we have k categories, we end up with k common logistic classifiers for different tags.
As with KNN, data needs to be reshaped and normalized before feeding the model features. The overall logistic regression code is shown in Figure 10.



```
def LR(x_train, y_train, x_test, y_test):
  x_train = x_train.reshape(60000, -1)
  x_test = x_test.reshape(10000, -1)
  # y_train = y_train[:1000]
  # y_test = y_test[:100]
  for i, line in enumerate(x_train):
    for j, cur in enumerate(line):
      if x_train[i][j] > 0:
        x_train[i][j] = 1
  for i, line in enumerate(x_test):
    for j, cur in enumerate(line):
      if x_test[i][j] > 0:
        x_test[i][j] = 1
  lr = LogisticRegression()
  lr.fit(x_train, y_train)
  result = lr.predict(x_test)
  print('LR accuracy=', accuracy_score(y_test, result))

if __name__ == '__main__':
  (x_train_image,y_train_label), (x_test_image,y_test_label) = mnist.load_data()
  # KNN(x_train_image, y_train_label, x_test_image, y_test_label)
  LR(x_train_image, y_train_label, x_test_image, y_test_label)
```

Figure 10 The overall logistic regression code

Support Vector Machine (SVM)

Algorithm Introduction

The full name of SVM is Support Vector Machine, which is an efficient way to solve the problem of data classification. A common SVM can be regarded as a straight line, which is used to perfectly classify two types of linearly separable, but it is not an ordinary straight line. It is the most perfect one of the numerous lines that can be classified, because it happens to be in the middle of two classes, as far as the points of the two classes. The Support Vector is the closest point to the dividing line. SVM is widely used in the field of data pattern recognition.

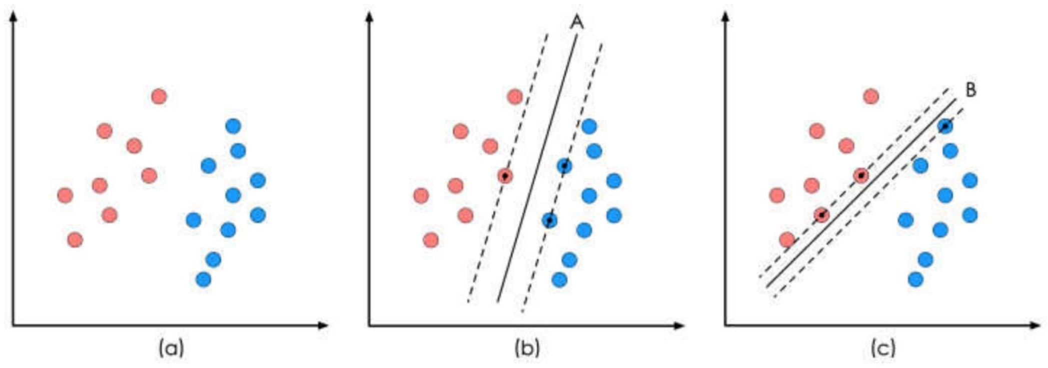
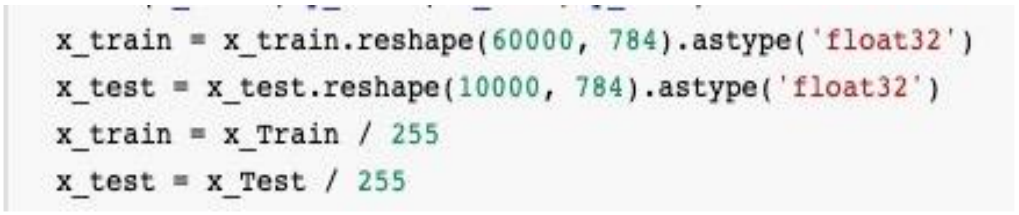


Figure 11 SVM solves the Binary-Classification problem

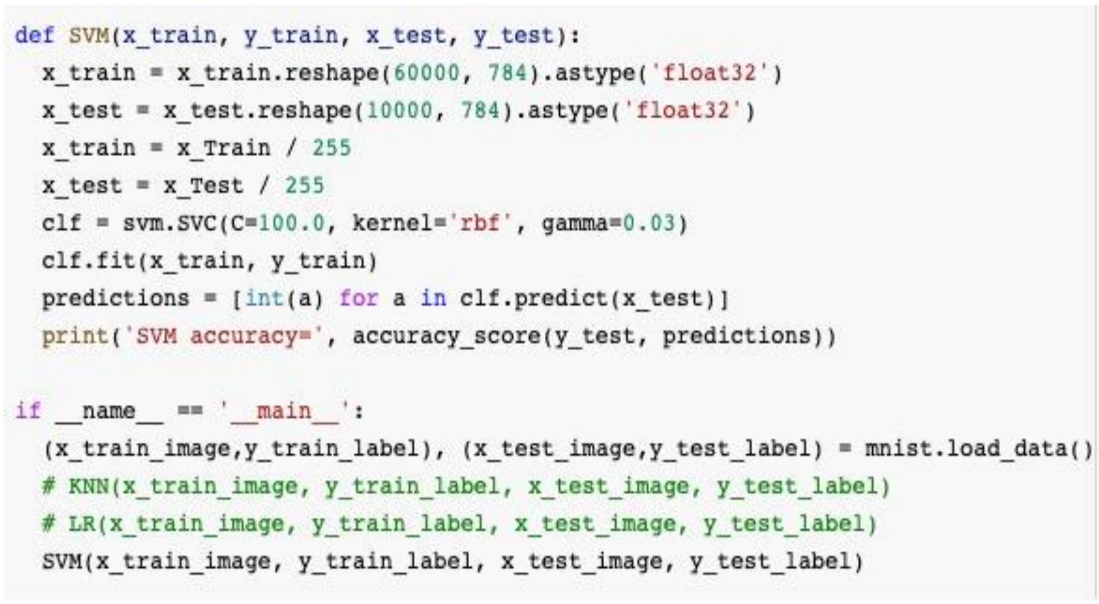
Algorithm Implementation

Similarly, SVM and LR can not only solve the Binary-Classification problem, but also expand into the classifier of multi-classification problem. However, unlike KNN and LR, SVM can use continuous numerical features as data instead of 0/. 1 discrete value as a feature input, which can fully describe the data set, so the data needs to be normalized first, the specific code is shown in Figure 12, the overall code of the SVM is shown in Figure 13.



```
x_train = x_train.reshape(60000, 784).astype('float32')
x_test = x_test.reshape(10000, 784).astype('float32')
x_train = x_Train / 255
x_test = x_Test / 255
```

Figure 12



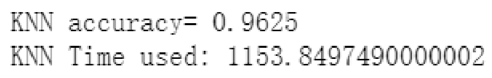
```
def SVM(x_train, y_train, x_test, y_test):
  x_train = x_train.reshape(60000, 784).astype('float32')
  x_test = x_test.reshape(10000, 784).astype('float32')
  x_train = x_Train / 255
  x_test = x_Test / 255
  clf = svm.SVC(C=100.0, kernel='rbf', gamma=0.03)
  clf.fit(x_train, y_train)
  predictions = [int(a) for a in clf.predict(x_test)]
  print('SVM accuracy=', accuracy_score(y_test, predictions))

if __name__ == '__main__':
  (x_train_image,y_train_label), (x_test_image,y_test_label) = mnist.load_data()
  # KNN(x_train_image, y_train_label, x_test_image, y_test_label)
  # LR(x_train_image, y_train_label, x_test_image, y_test_label)
  SVM(x_train_image, y_train_label, x_test_image, y_test_label)
```

Figure 13

Evaluation

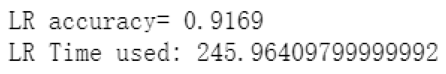
Report Execution on Data and Perform Efficiency Analysis

We performed experiments on the mnist dataset using the three algorithms mentioned above on the colab platform. The results are shown in Figures 14 15 and 16 The accuracy of the KNN algorithm is 96.25%. The accuracy rate is 91.69%, and the accuracy of the SVM algorithm is 98.57%.
We used the above mentioned three algorithms on the colab platform to experiment on the mnist dataset and recorded the running time. The machine configuration of colab is not high, so these three algorithms run longer but meaningful, the running time of the KNN algorithm is 1153.84 seconds, the running time of the LR algorithm is 245.96 seconds, the running time of the SVM is 3774.16 seconds.



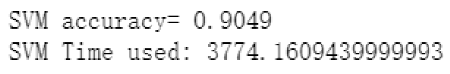
```
KNN accuracy= 0.9625
KNN Time used: 1153.8497490000002
```

Figure 14



```
LR accuracy= 0.9169
LR Time used: 245.96409799999992
```

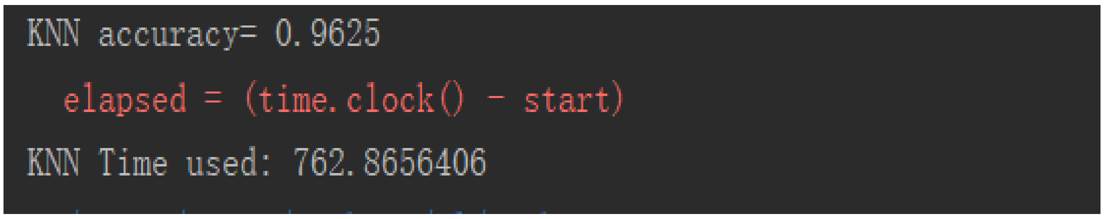
Figure 15



```
SVM accuracy= 0.9049
SVM Time used: 3774.1609439999993
```

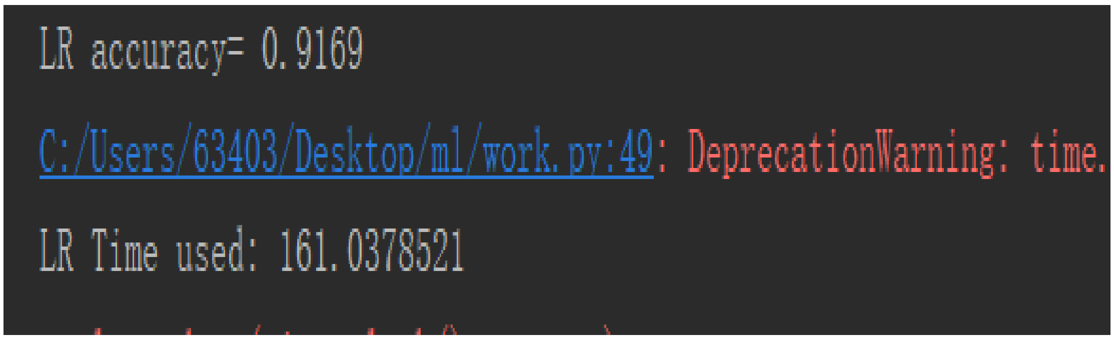
Figure 16

If we run the code in the pycharm environment, the runtime will increase slightly and the accuracy will not change, as shown in Figure 17， 18， 19.



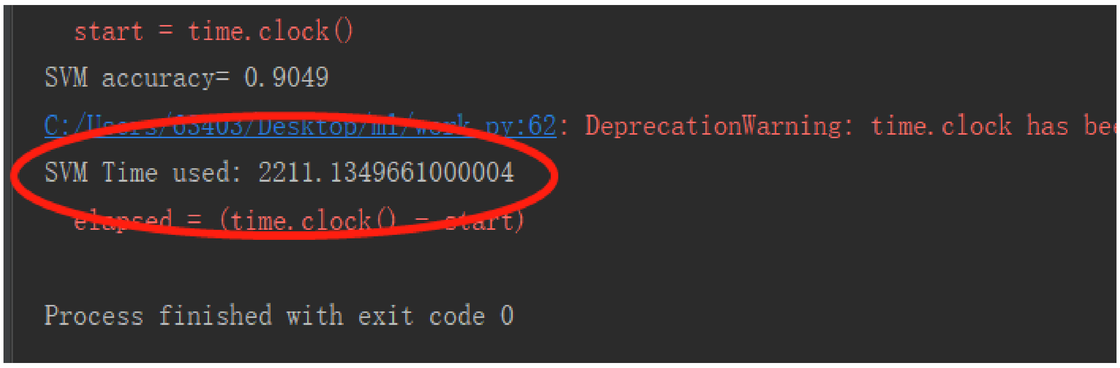
```
KNN accuracy= 0.9625
  elapsed = (time.clock() - start)
KNN Time used: 762.8656406
```

Figure 17



```
LR accuracy= 0.9169
C:/Users/63403/Desktop/ml/work.py:49: DeprecationWarning: time.
LR Time used: 161.0378521
```

Figure 18



```
  start = time.clock()
SVM accuracy= 0.9049
C:/Users/63403/Desktop/ml/work.py:62: DeprecationWarning: time.clock has be
SVM Time used: 2211.1349661000004
  elapsed = (time.clock() - start)

Process finished with exit code 0
```

Figure 19

Comparative Study

From the results, we can see that SVM works best. KNN is the second, LR result is the worst. The reason is that SVM is a nonlinear classifier, which can learn the relationship between features well, and LR is a linear classifier. To the connection between features, only the weight of a single feature can be learned. Although the KNN algorithm is a model-like algorithm similar to the rule, mnist is more suitable for the KNN algorithm, but since K needs to be artificially set, the parameter adjustment is cumbersome.
Although the SVM works best, it has been found through analysis that it takes the longest time, so it is necessary to choose which algorithm to choose in the actual application, and also need to balance the time-consuming and accurate rate.

Conclusion

Discuss Reflections

In this paper, three machine learning algorithms are applied to the mnist dataset in the colab environment. It is found that the SVM algorithm performs best, but it takes the most time. The LR algorithm should be very bad because it is more suitable for two-category tasks, such as click-through rate estimation tasks. Relatively speaking, KNN is relatively good at the accuracy of timeliness.

Propose Possible Improvements

Nowadays, the development of deep learning technology is in full swing. It has achieved good results in computer vision and natural language processing tasks. The deep neural network is a relatively simple deep learning network. In theory, it can fit any nonlinear function, and finally use the softmax function. Do more classification, so it can also be applied to the mnist data set, and its network structure is shown in Figure 20.

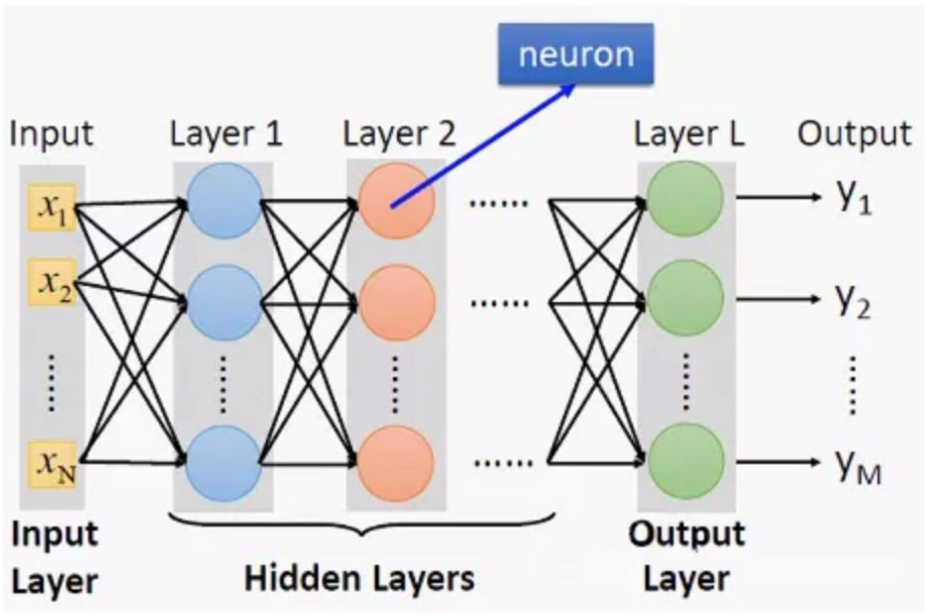


Figure 20 Deep Neural Network

Convolutional Neural Networks (CNNs / ConvNets) is more suitable for the learning of picture recognition tasks. A classic LeNet-5 network result is shown in Figure 21. From the relevant article, the convolutional neural network technology is applied in mnist. Nearly 100% accuracy can be obtained on the data set.

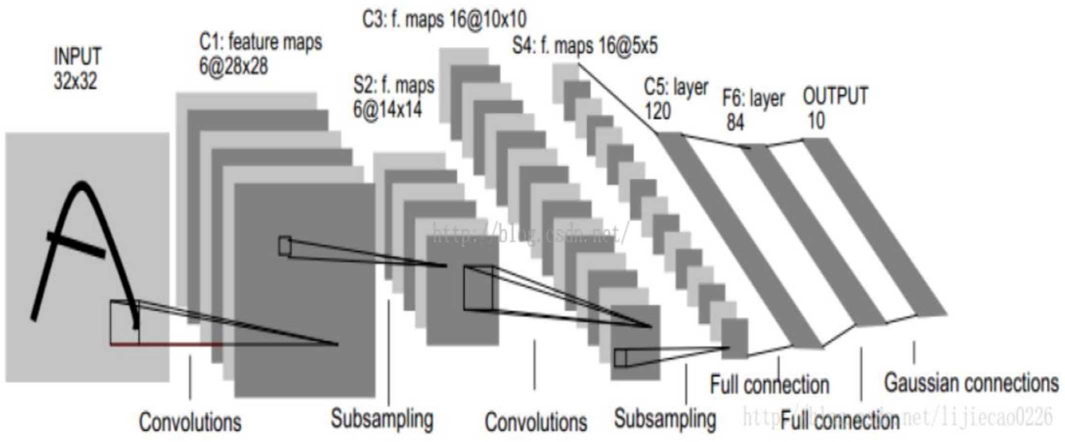


Figure 21 LeNet-5 convolutional neural network structure

In addition, there are many more complex deep neural networks, such as the Alexnet network in Figure 22, and the deeper ResNet network in Figure 23. The accuracy of the mnist data level has been brushed to a high level. Now evaluate the deep learning technology. The data set has been extended from grayscale handwritten digits to color multi-scene multi-material identification.

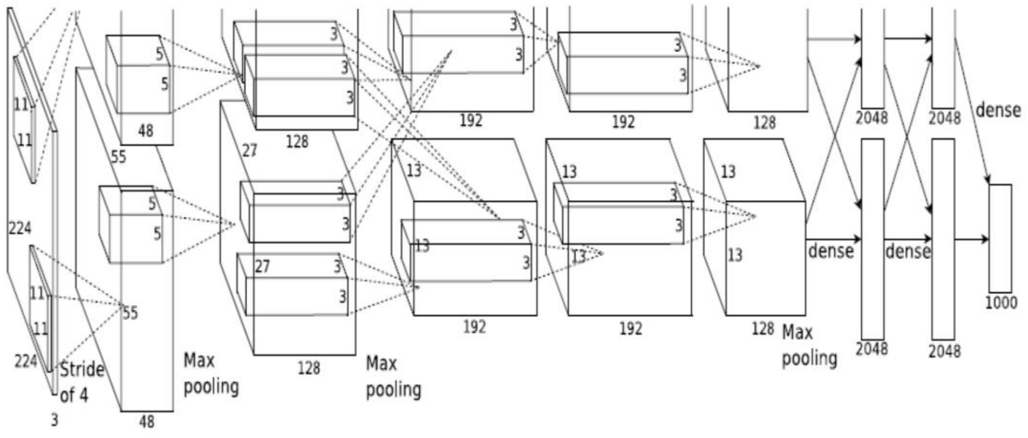


Figure 22 Alexnet Network

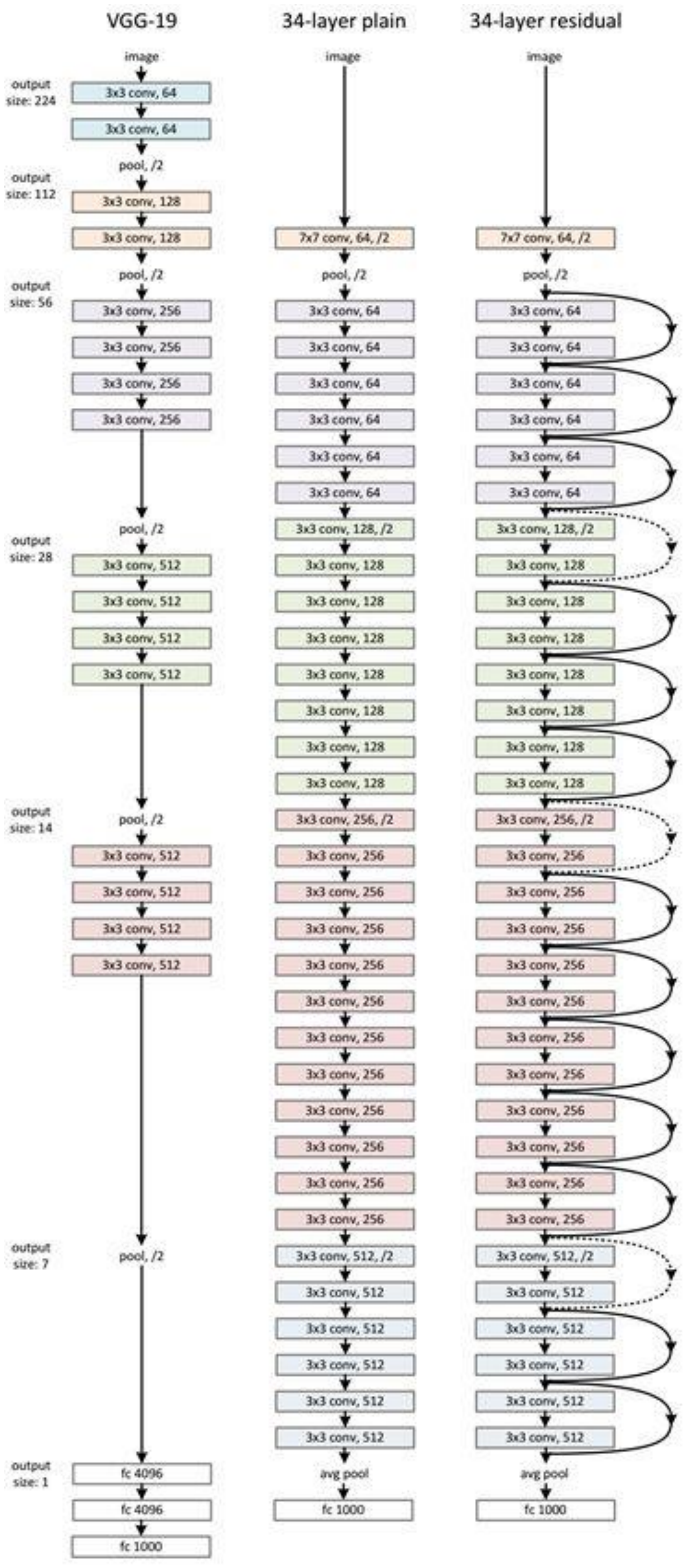


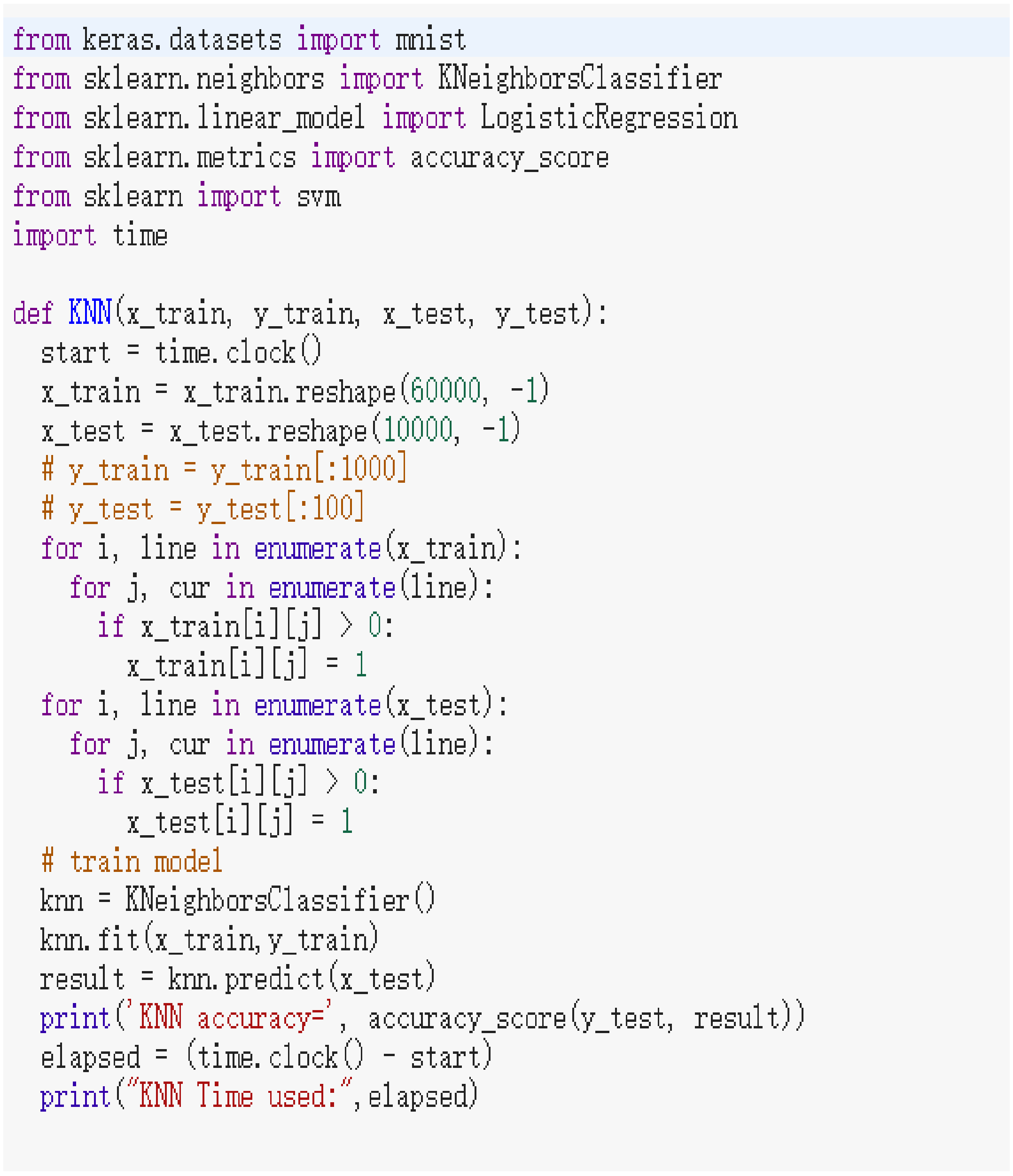
Figure 23 ResNet Network

Ethical

In 2018, Microsoft published The Future Computed, which contains six principles for AI development: fairness, reliability and security, privacy and security, inclusiveness, transparency and responsibility. The first is fairness. Fairness means that no one should be discriminated against. For people, no matter what region or level of people, all people have no particularity in front of AI. All people are equal. The second is reliability and security. It means that AI is safe, reliable and not evil to use. The third is privacy and protection. Because AI involves data, it always causes problems in personal privacy and data security. Fourthly, AI must take into account the inclusive moral principles and the various dysfunctional groups in the world. The fifth is transparency. In the past decade, the most important technology in the field of artificial intelligence has made great progress. Deep learning is a model of machine learning. We believe that at least at this stage, the accuracy of deep learning model is the highest among all machine learning models, but there is a question whether it is transparent or not. Transparency and accuracy can't be achieved at the same time. You can only trade off between them. If you want higher accuracy, you have to sacrifice a certain degree of transparency. The sixth is accountability. The AI system takes an action and makes a decision. It must be responsible for the results it brings. It can be seen from this paper that machine learning algorithm can get high accuracy for handwritten numeral recognition tasks. Although there are many practical scenarios that can land and bring convenience to human beings, we should pay attention to the ethical nature of AI while using these technologies, so as not to do evil.

Code：

The entire code of assignment 2:



```python
from keras.datasets import mnist
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import svm
import time

def KNN(x_train, y_train, x_test, y_test):
  start = time.clock()
  x_train = x_train.reshape(60000, -1)
  x_test = x_test.reshape(10000, -1)
  # y_train = y_train[:1000]
  # y_test = y_test[:100]
  for i, line in enumerate(x_train):
    for j, cur in enumerate(line):
      if x_train[i][j] > 0:
        x_train[i][j] = 1
  for i, line in enumerate(x_test):
    for j, cur in enumerate(line):
      if x_test[i][j] > 0:
        x_test[i][j] = 1
  # train model
  knn = KNeighborsClassifier()
  knn.fit(x_train,y_train)
  result = knn.predict(x_test)
  print('KNN accuracy=', accuracy_score(y_test, result))
  elapsed = (time.clock() - start)
  print(“KNN Time used:”,elapsed)
```

```
def LR(x_train, y_train, x_test, y_test):
  start = time.clock()
  x_train = x_train.reshape(60000, -1)
  x_test = x_test.reshape(10000, -1)
  # y_train = y_train[:1000]
  # y_test = y_test[:100]
  for i, line in enumerate(x_train):
    for j, cur in enumerate(line):
      if x_train[i][j] > 0:
        x_train[i][j] = 1
  for i, line in enumerate(x_test):
    for j, cur in enumerate(line):
      if x_test[i][j] > 0:
        x_test[i][j] = 1
  lr = LogisticRegression()
  lr.fit(x_train, y_train)
  result = lr.predict(x_test)
  print('LR accuracy=', accuracy_score(y_test, result))
  elapsed = (time.clock() - start)
  print("LR Time used:",elapsed)

def SVM(x_train, y_train, x_test, y_test):
  start = time.clock()
  x_train = x_train.reshape(60000, 784).astype('float32')
  x_test = x_test.reshape(10000, 784).astype('float32')
  x_train = x_train / 255
  x_test = x_test / 255
  clf = svm.SVC(C=100.0, kernel='rbf', gamma=0.03)
  clf.fit(x_train, y_train)
  predictions = [int(a) for a in clf.predict(x_test)]
  print('SVM accuracy=', accuracy_score(y_test, predictions))
  elapsed = (time.clock() - start)
  print("SVM Time used:",elapsed)

if __name__ == '__main__':
  (x_train_image,y_train_label), (x_test_image,y_test_label) = mnist.load_data()
  KNN(x_train_image, y_train_label, x_test_image, y_test_label)
  LR(x_train_image, y_train_label, x_test_image, y_test_label)
  SVM(x_train_image, y_train_label, x_test_image, y_test_label)
```

Reference

Ayani, R. 1990, 'Lr-algorithm: concurrent operations on priority queues', Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing 1990, IEEE, pp. 22-5.

Foody, G.M. & Mathur, A. 2004, 'Toward intelligent training of supervised image classifications: directing training data acquisition for SVM classification', Remote Sensing of Environment, vol. 93, no. 1-2, pp. 107-17.

Moor, J.H. 2006, 'The nature, importance, and difficulty of machine ethics', IEEE intelligent systems, vol. 21, no. 4, pp. 18-21.

Moreno, P.J., Ho, P.P. & Vasconcelos, N. 2004, 'A Kullback-Leibler divergence based kernel for SVM classification in multimedia applications', Advances in neural information processing systems, pp. 1385-92.

Zhang, H., Berg, A.C., Maire, M. & Malik, J. 2006, 'SVM-KNN: Discriminative nearest neighbor classification for visual category recognition', 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), vol. 2, IEEE, pp. 2126-36.