Digital recognization by five methods

Lanhe Gao 2018533212
SIST
ShanghaiTech University
Shanghai, China
`gaolh@shanghaitech.edu.cn`

Yunfei Zhang 2018533098
SIST
ShanghaiTech University
Shanghai, China
`zhanyf2@shanghaitech.edu.cn`

Abstract

Recognizing handwritten numbers is a very classical machine learning problem, which has a variety of solutions. In this report, we try to implement five of these methods. By comparing their performances, we hope to choose the best one.

1 Introduction

Number recognization is a widely used application, with great sophistication. There are many types of machine learning methods that can be used in this scenario. In this report, we have adopted five methods, such as Support Vector Machine (SVM), K-Nearsest Neighbors (KNN), Naive Bayes (NB), Neural Networks (NN) and Logistic Regression (LR). We will analyze the accuracy, runtime performance, confusion matrix and other standards to compare their performance, and conclude the best method.

2 Data Setting and Methods

2.1 Data setting

Our training data and testing data are collected from *mnist* dataset. They're in the same form, with 28*28 pixels, therefore, we use a 784 dimensional vector to represent each number picture. The pictures are gray images, scaling from 0 to 255.

2.2 KNN

The first method we use is KNN. Consider that each picture is composed of 784 pixels. If we expand it into a 784-dimensional vector in a fixed spatial order, we can put each test point into a hyperplane. Since the dimension is quite big, which makes the complexity of calculating 2-norm quite big, so we use 1-norm instead. KNN calculates K nearest images that has the smalles 1-norm distance to a given test data, and labels it as the class that contains the most number of neighbors.

It takes at least $O(784)$ time to calculate the distance for every two pictures. The number of samples in the training set is 60,000, the number of samples in the test set is 10,000, and the time responsibility is $O(5 * 10^1 1)$, which is too long. In our test, we select the first 100 test set samples for testing, which approximates the matching rate of the KNN algorithm.

The code implementation is at

`https://github.com/chuansao-258/machine-learning--digital-recognition/blob/main/knn.ipynb`

Preprint. Under review.

2.3 SVM

The SVM algorithm was originally designed for binary classification problems. When dealing with multi-class problems, it is necessary to construct a suitable multi-class classifier. So when extracting the training set, take class 0 as an example, extract:

$$+ : 0, - : 1 \sim 8 \tag{1}$$

Other class work similarly. When testing, use the ten training result files to test the corresponding test vectors. Finally, each test has a result $f_0(x), f_1(x), \cdots, f_9(x)$. The final result is the largest of these four values as the classification result.

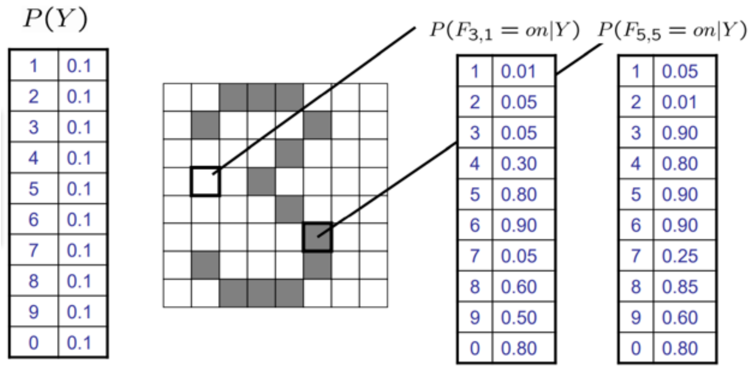
We use the svm from *sklearn* package. We directly put the data and label into the model *svm.fit()* for training, then use *svm.predict()* to run prediction. The code implementation is at

`https://github.com/chuansao-258/machine-learning--digital-recognition/blob/main/SVM.ipynb`

2.4 NB

In Bayes model. we assume all features are independent effects of the label. Simple digit recognition version: One feature(variable) $F_{i,j}$ for each grid position $< i, j >$. Feature values are on/off, based on whether intensity is more or less than 0.5 in underlying image. Each input maps to a feature vector, e.g. Image1 $\rightarrow (F_{0,0} = 0, F_{0,1} = 0, F_{0,2} = 1, ..., F_{27,27} = 0)$.

Then, the prediction works like this: given a test image, we calculate the probability multiplication of all pixel points, and choose the label with the highest probability, As shown in Figure 1.



Y	$P(Y)$	$P(F_{3,1} = on\|Y)$	$P(F_{5,5} = on\|Y)$
1	0.1	0.01	0.05
2	0.1	0.05	0.01
3	0.1	0.05	0.90
4	0.1	0.30	0.80
5	0.1	0.80	0.90
6	0.1	0.90	0.90
7	0.1	0.05	0.25
8	0.1	0.60	0.85
9	0.1	0.50	0.60
0	0.1	0.80	0.80

Figure 1: Bayes classification

Before coding, we tried the naive bayes model in sklearn. However, it does not work well, with only 58% accuracy, so we try to implement one by ourselves. The code implementation is at

`https://github.com/chuansao-258/machine-learning--digital-recognition/blob/main/bayes.ipynb`

2.5 LR

Similar to SVM, Logistic Regression is also a binary classification method. However, we want to classify these handwritings into 10 classes, so we need to extend LR method as well. For each class, we devide the dataset into the ones belong to the class, and the ones do not. For example, from class 0's perspective, the dataset would be labeled as:

$$+ : 0, - : 1 \sim 8 \tag{2}$$

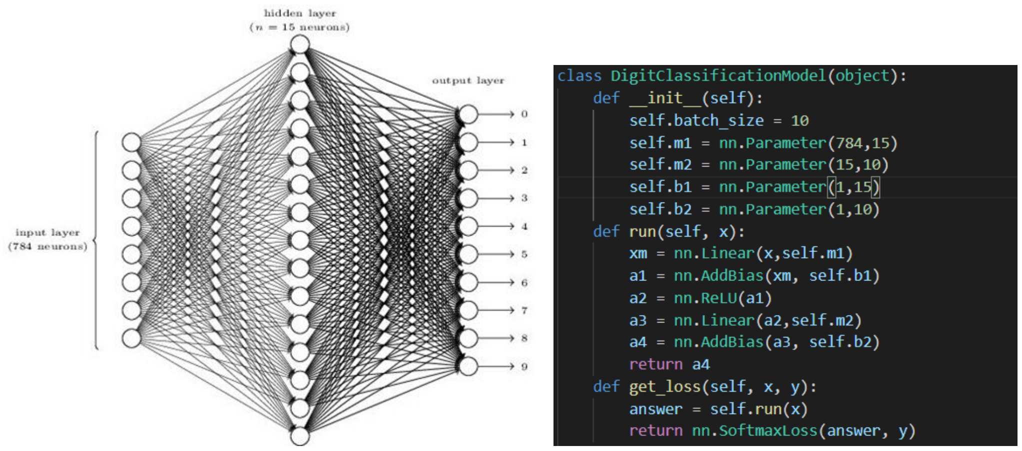
This turns out to be a binary classification problem now, and we need to do it for every class, 10 times in total. Finally, we choose the class that the input has the highest probability on.

We use the LogisticRegression model in sklearn to implement this method. The code implementation is at

`https://github.com/chuansao-258/machine-learning--digital-recognition/blob/main/Logistic%20Regression.ipynb`

2.6 NN

The settings of the neural network is displayed as Figure 2(a) and 2(b). The first 784 layers enter the neural network as input. The 15th layer is used as the middle layer; after passing through the middle layer, it enters the activation function 10 layers as the output layer, representing the possible probability of each picture for each number. Choose the largest one as the prediction result, and use the gradient descent method to modify the values of m1, b1, m2, and b2.



```python
class DigitClassificationModel(object):
    def __init__(self):
        self.batch_size = 10
        self.m1 = nn.Parameter(784,15)
        self.m2 = nn.Parameter(15,10)
        self.b1 = nn.Parameter(1,15)
        self.b2 = nn.Parameter(1,10)
    def run(self, x):
        xm = nn.Linear(x,self.m1)
        a1 = nn.AddBias(xm, self.b1)
        a2 = nn.ReLU(a1)
        a3 = nn.Linear(a2,self.m2)
        a4 = nn.AddBias(a3, self.b2)
        return a4
    def get_loss(self, x, y):
        answer = self.run(x)
        return nn.SoftmaxLoss(answer, y)
```

Figure 2: Neural network setting

The code implementation is at

`https://github.com/chuansao-258/machine-learning--digital-recognition/tree/main/nn`

3 Performance

All the performances can be checked in the code implementation.

3.1 KNN

The parameters of KNN we set is:

- Manhattan distance
- k = 5

And the accuracy is 47.00%. This method takes long time to calculate and the performance is not good.

3.2 NB

To prevent the case that the probability might fall to 0, we set $0 = 1e^{-}7$. The accuracy is 68.41%

3.3 LR

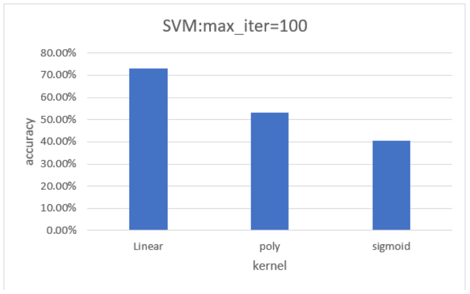
We set the solver in LogisticRegression model in sklearn as 'saga', and the accuracy is 92.57%. This method can be seen as neural network without hidden layer, so we expect higher accuracy in NN method.

3.4 SVM

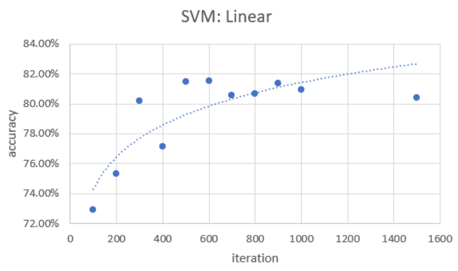
In the SVM model, there are two factors that might affect the accuracy,

- max_iter: the number of iterations.
- kernel: the kernel function type. the choices are 'linear', 'poly' and 'sigmoid'.

We've adjusted these two and the results are shown in Figure 3(a) and 3(b).



(a) accuracy-kernel



(b) accuracy-max_iter

Figure 3: SVM performance of two factors

We can see that linear kernel with high iteration number, the accuracy could be very high, about 82%.

3.5 NN

In NN model, we find three factors that might affect the accuacy,

- layer1: the number of units in the hidden layer. the choices are 15 or 100.
- batch_size: the number of training samples to choose in each round of training. the choices are 10 and 100.
- iteration: the number of iterations in training.

We've compared six groups of parameters, and the performance is shown in Figure 4. Apparently, the set $[100, 10, 30]$ works the best, with about 97.94%

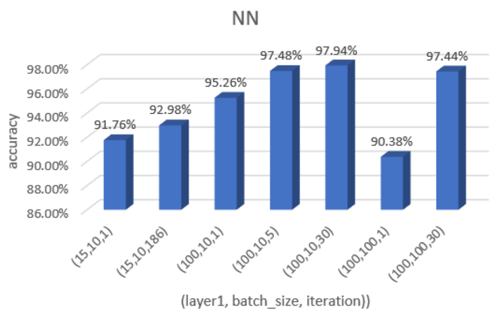


Figure 4: NN performance of three factors

4 Conclusion

After impleenting these five methods, we can conclude that neural network works the best, with an astonishing 97.94% accuracy. However, there are other choices, like convolutional neural network, recursive neural network, decision tree and so on. We hope to develop their possiblities in achieving higher accuracy in further research and implementation.