Journal of Multidisciplinary Engineering Science and Technology (JMEST)
ISSN: 2458-9403
Vol. 7 Issue 9, September - 2020

Research and Implementation of Iris Classification Algorithm based on Neural Network

Bingxing Yu, Hongli Zhu
School of Information and Electronic Engineering
Zhejiang University City College, Hangzhou, China
Corresponding Author: zhuhl@zucc.edu.cn

***Abstract*— This paper introduces the classification of iris by building BP neural network model. First of all, a three-layer neural network model with reasonable structure is constructed; then 150 original iris datasets are randomly scrambled and divided into 100 training sets and 50 test sets; then 100 data are put into the model for training to get the appropriate weight value; finally, through the test set test experiment results, the data accuracy is obtained. Finally, the data show that the accuracy of this method can reach more than 90%, which proves that the neural network model has a high accuracy in the classification algorithm. At the same time, the advantages and disadvantages of BP neural network are compared with other machine learning algorithms.**

***Keywords*— Irises；machine learning；neural networks**

I. INTRODUCTION

In the 1980s, the performance of neural networks at that time was not outstanding, but the discovery of a backpropagation algorithm brought new vitality to neural networks. This algorithm also gave detailed derivations in the field of mathematics. In the process, people call the multi-layer feedforward network using this algorithm as a BP network. The discovery of this method systematically solves the problem of connection learning between hidden layers of many multilayer neural networks, which is called BP neural network.

BP neural network is proposed based on the principle of human brain nerves responding to external stimuli. It is a multi-layer feedforward network. The BP algorithm is an algorithm that uses the gradient descent method to calculate the minimum target value for the purpose of minimizing the mean square error of the error. It has excellent nonlinear mapping ability and outstanding classification ability. It is the proposal of its model that solves most of the problems that can not be solved by simple components and realizes more comprehensive functions.

Although the BP neural network has excellent non-linear mapping capabilities and powerful classification capabilities, its overly strong self-learning and adaptive capabilities make it also have some difficult problems to solve. With in-depth research, these problems have also been recognized. One proposed: 1. Excessive self-learning ability can easily lead to its local optimal; 2. Over-fitting phenomenon between detection ability and training ability is easy to appear 3. Too complicated structure makes it need more time in the training process to calculate and adjust, resulting in too slow convergence.

II. DATA PREPROCESSING

Algorithm assumption: The algorithm is mainly composed of three aspects, which are data preprocessing, BP neural network realization and the presentation of the final result.

Data preprocessing: The 150 iris data sets used in this article need to be processed to store the feature values and their labels separately for convenience and subsequent calculation and verification. At the same time, the data needs to be randomly allocated to complete the subsequent training and detection work. As the first data preprocessing work to be achieved, it is particularly important. If the steps of data segmentation and conversion are improperly operated, it will directly affect the operation of the entire algorithm and even cause it to fail to run correctly.

BP neural network realization: This aspect is mainly divided into two steps, namely data training and result verification. During training, the BP neural network needs to be carried out with the support of many computing components. After defining each important component, the pre-allocated training data set will be processed according to the operation process described in Chapter 2, so as to realize the final algorithm and obtain each The result of the weight value. During the test, the pre-allocated test data set is predicted by a built algorithm model, and the correct rate of the final prediction is obtained by comparing the original label.

Result presentation: The calculation of each weight value in the neural network algorithm is a very important process, so it is necessary to show the specific value of each weight value in the model after the result is run. At the same time, the accuracy of model prediction is also a very important indicator. Show it in the final output.

According to the above three specific steps to implement the specific code of the algorithm, the

algorithm can be realized. See below for specific code and function of the algorithm.

The following is the preliminary data preprocessing work.

Data reading

This method reads a local file, or you can use the iris data set that can be directly called in sklearn, and the read content is the same

```
raw = pd.read_csv(r'C:\Users\apple\Desktop\iris.csv')
```

Read the data value in the file (the content of the first four columns)

```
raw_data = raw.values
```

Save the data values and classification labels in the file to two dimensions respectively to facilitate subsequent one-to-one correspondence

```
raw_feature = raw_data[0:, 0:4]
```

Data category conversion

Convert the iris category in the last column to one-hot encoding (simplify category storage format, improve learning efficiency), the three categories correspond to: setosa corresponds to [1, 0, 0]; versicolor' corresponds to [0, 1, 0]; viegincia corresponds to [0, 0, 1]

```
for i in range(len(raw_feature)):
    fenlei = []
    fenlei.append(list(raw_feature[i]))
    if raw_data[i][4] =='Iris-setosa':
        fenlei.append([1, 0, 0])
    elif raw_data[i][4] =='Iris-versicolor':
        fenlei.append([0, 1, 0])
    else:
        fenlei.append([0, 0, 1])
  data.append(fenlei)
```

Data segmentation

Because the follow-up work requires a training set to determine the specific structure of the model and a test set to detect the correct rate of the resulting model, the original 150 data need to be segmented

Random scramble data

```
numpy.random.shuffle(data)
```

Select the first 100 scrambled as the training data (the first value (0) is not taken, the last value (100) is taken):

```
training = data[0:100]
```

Select the last 50 scrambled as test data:

```
test = data[100:]
```

In the process of data preprocessing, the statement numpy.random.shuffle(data) is particularly important, which will have a great impact on subsequent results. When using random.shuffle( ) originally, the results have a great impact. The accuracy of the classification results has not changed, and the data classifications obtained are all in the same array, which does not achieve the original desired effect. The reason is that random allocation of arrays by random shuffle may not be implemented well, sometimes it is good and sometimes bad, and the result of random allocation for iris data cannot be achieved, so the final output result does not change, while shuffle in numpy. This problem can be solved very well, so the function in numpy is needed to randomly classify the array.

III. ALGORITHM

Initial definition:

```
def __init__(self, shuru, yinhan, shuchu):
# shuru is the node of the input layer, yinhan is the node of the hidden layer, and shuchu is the node of the output layer
        self.shuru = shuru + 1
        self.yinhan = yinhan + 1
        self.shuchu = shuchu
#Activate all nodes of the neural network (convert all of them to matrix form to facilitate subsequent calculations)
        self.srjd = [1.0] * self.shuru
        self.yhjd = [1.0] * self.yinhan
        self.scjd = [1.0] * self.shuchu
# Build a matrix to store all weight values
        self.srqzz = juzhen(self.shuru, self.yinhan)
        self.yhqzz = juzhen(self.yinhan, self.shuchu)
# Set the weight value to a random value
         for i in range(self.shuru):
            for j in range(self.yinhan):
                self.srqzz[i][j] = rand(-0.2, 0.2)
        for j in range(self.yinhan):
            for k in range(self.shuchu):
                self.yhqzz[j][k] = rand(-2, 2)
```

*Generate a matrix of size I*J, the default zero matrix is used to store each weight value*

```
def juzhen(I, J, fill=0.0):
    m = []
    for i in range(I):
        m.append([fill] * J)
    return m
```

The sigmoid function is used as an activation function

```
    def sigmoid(x:
    return 1.0 / (1.0 + math.exp(-x))
```

The derivative of the sigmoid function, used for back propagation calculation

```
def dsigmoid(x):
  return x * (1-x)
```

Output layer error calculation and output layer weight value update

```
#Calculate the error of the output layer:
scwc = [0.0] * self.shuchu
      for k in range(self.shuchu):
         error = targets[k]-self.scjd[k]
         scwc[k] = dsigmoid(self.scjd[k]) * error
#Update output layer weight:
for j in range(self.yinhan):
         for k in range(self.shuchu):
            change = scwc[k] * self.yhjd[j]
            self.yhqzz[j][k] = self.yhqzz[j][k] + lr * change
```

Hidden layer error calculation and hidden layer weight value update

```
#Calculate the error of the hidden layer:
    yhwc = [0.0] * self.yinhan
      for j in range(self.yinhan):
         error = 0.0
         for k in range(self.shuchu):
            error = error + scwc[k] * self.yhqzz[j][k]
         yhwc[j] = dsigmoid(self.yhjd[j]) * error
#Update hidden layer weight:
    for i in range(self.shuru):
         for j in range(self.yinhan):
            change = yhwc[j] * self.srjd[i]
            self.srqzz[i][j] = self.srqzz[i][j] + lr * change
```

Weight value display

```
#Output the updated weight value in real time during the training process, making the internal parameters of the network structure model more intuitive
def weights(self):
      print('Input layer weight:')
      for i in range(self.ni):
         print(self.wi[i])
      print()
      print('Output layer weight:')
      for j in range(self.nh):
         print(self.wo[j])
```

Training process

```
#The training process defaults to 1000 times, and the learning rate defaults to 0.1
def train(self, patterns, iterations=1000, lr=0.1):
      for i in range(iterations):
         error = 0.0
         for p in patterns:
            inputs = p[0]
            targets = p[1]
            self.update(inputs)
            error = error + self.backPropagate(targets, lr)
         if i% 100 == 0:
            print('error: %-.9f'% error)
self.weights()
```

Test process

```
def test(self, patterns):
      count = 0
      for p in patterns:
         target = flowerLables[(p[1].index(1))]
         result = self.update(p[0])
         index = result.index(max(result))
         print(p[0],':', target,'->', flowerLables[index])
         count += (target == flowerLables[index])
      accuracy = float(count / len(patterns))
#Output correct rate
      print('accuracy: %-.9f'% accuracy)
```

IV. EXPERIMENTAL RESULTS

It can be seen from the constructed BP neural network model that the weight value of the input layer is a matrix of 5 rows and 8 columns, and 4 input data plus 1 deviation node (5 nodes in total) correspond to 7 hidden layer nodes plus 1 deviation node (total 8 nodes); the hidden layer weight value is a matrix of 8 rows and three columns, 7 hidden layer nodes plus 1 deviation node (total 8 nodes) corresponding to three output layer nodes, as shown in Figure 4.1. The values are all correct. The test result is the data in the 50 test data sets that we have pre-allocated. Figure 4.2 shows that the test set is allocated correctly.

The training results show that under good parameter conditions, the method can classify the correct rate of 94% or even higher in the 150 iris data set, achieving the expected training effect, indicating that the BP neural network has a high accuracy rate.

REFERENCES

[1] Guo G ,Wang H , Bell D A ,et al. KNN Model-Based Approach in Classification[C]. On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Springer, Berlin, Heidelberg, 2003.

[2] Lecun Y,Bengio Y,Hinton G. Deep learning.[J]. Nature.2015,521(7553):436.

[3] Tang Yuzheng. Discriminant analysis based on Euclidean distance: A Study on the classification of iris [J]. Modern commerce and industry. 2019,40 (9): 187-189

[4] Zhou Qingping. Research on Improved BP neural network algorithm based on rough set [J]. Modern industrial economy and informatization. 2015 (5): 87-88

[5] Lu Dunli, Ning Qian, Zang Jun. improved KNN algorithm based on BP neural network decision making [J]. Computer application. 2017,37 (S2): 65-67