Classifying Congressional Voting Records Data Set: Implementation A Simple Artificial Neuron Network And Comparing With Other Works

Author Name (Paper 21 assignment 2)
Research School of Computer Science, Australian National University
u1234567@anu.edu.au

Abstract:
Neural network related to deep learning is significant in modern society, with more methods and researches are developed and applied. This paper will mainly discuss about an artificial neural network implemented based on a real-world dataset to solve a classification problem with PyTorch / Python code. Performances of the neural network will be observed and compared with another neural network focusing on the same problem but improved with a technique called K-fold cross-validation. The performance of the improved network is better, however, still worse than another research paper which had used on the same dataset. Reasons of the difference will be discussed and some future work will be mentioned.

Keywords: Neural Network, K-fold cross-validation, Real World Data Set, Pre-processing

1. Introduction:

The Neural network applied in the paper is designed to solve a classification problem with 16 inputs (attributes) and 2 outputs (targets). The dataset for this problem is a real-world data set about congressional voting records, with 16 categorical attributes for classification problem. The website for the dataset is: http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records
Figure 1-1 shows the details description for the dataset:

Data Set Characteristics:	Multivariate	**Number of Instances:**	435	**Area:**	Social
Attribute Characteristics:	Categorical	**Number of Attributes:**	16	**Date Donated**	1987-04-27
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	138845

Figure 1-1: details for the dataset

The dataset has the following attributes (Figure 1-2), and the artificial neural network designed will count those attributes from index 2 to index 17 (handicapped-infants, water-project-cost-sharing, etc. 16 attributes in total) in figure 1-2 as input and the "Class Name" (index 1 in figure 1-2) as output (classification type/target). So with 16 inputs, a desired output (either: democrat or republican) will be presented by the well-trained neural network.

1. Class Name: 2 (democrat, republican)
2. handicapped-infants: 2 (y,n)
3. water-project-cost-sharing: 2 (y,n)
4. adoption-of-the-budget-resolution: 2 (y,n)
5. physician-fee-freeze: 2 (y,n)
6. el-salvador-aid: 2 (y,n)
7. religious-groups-in-schools: 2 (y,n)
8. anti-satellite-test-ban: 2 (y,n)
9. aid-to-nicaraguan-contras: 2 (y,n)
10. mx-missile: 2 (y,n)
11. immigration: 2 (y,n)
12. synfuels-corporation-cutback: 2 (y,n)
13. education-spending: 2 (y,n)
14. superfund-right-to-sue: 2 (y,n)
15. crime: 2 (y,n)
16. duty-free-exports: 2 (y,n)
17. export-administration-act-south-africa: 2 (y,n)

Figure 1-2: details of attributes

The reason why I choose this data set is that it is a classification problem and with enough attributes as well as instances. And also because the problem set is related with social area, which I am interested in and is really significant in modern society.

A considerable problem with the data set described above is that it has some missing values which will affect the behavior of accuracy while training and testing neural network. So how to solve the missing values is an important part of applying the neural network to solve the classification problem as described above.

To deal with the classification problem with an improved neural network, first step is to modify the raw data from the Congressional Voting Records data set. A simple neural network with just one input layer, one hidden layer and one output layer will then be created and different parameters will be tested until the output accuracy is good enough for a simple neural network. After that, a k-fold cross-validation will be applied to the original network, and the performance of the new network will be compared with the old one, to indicate if this method has been chosen is suitable for the original network and the data set. Also a related paper using the same data set will be discussed and the performance will be compared to indicate how to solve a problem with a same data set in a better way and how to further improve the performance of a neural network with a certain data set.

2. Method:

2.1 Pre-processing of data set:

The details and content of the Congressional Voting Records data set is described in figure 1-1 and figure 1-2, and all attributes are categorical values (only 'y' for 'yes' and 'n' for 'no'). The target is also a categorical type with only 'democrat' and 'republican', thus both inputs and outputs for training data and testing data should be first transform into countable values for applying in neural network. As a result, when reading the data into the program of neural network, all "republican" string values are changed into an integer value "0", and all "democrat" are turned into "2". For those attributes from index 2 to 17 in figure 1-2, all "y" are read as integer "7", while "1" for "n" and "4" for "?" ("?" stand for missing values in the raw data set). The reason why I change input attributes in this way is that "y" and "n" should be easy to distinguish in the network while "7" is much bigger than "1" and will give much stronger comment when training. As for missing value "?", it is the average value of "y" and "n", because an average value can give less information while training. Figure 2-1-1 shows an example

section of raw data set and figure 2-1-2 shows how does the section looks like after the pre-processing described above.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	republican	n	y	n	y	y	y	n	n	n	y	?	y	y	y	n	y
2	republican	n	y	n	y	y	y	n	n	n	n	n	y	y	y	n	?
3	democrat	?	y	y	?	y	y	n	n	n	n	y	n	y	y	n	n
4	democrat	n	y	y	n	?	y	n	n	n	n	y	n	y	n	n	y
5	democrat	y	y	y	n	y	y	n	n	n	n	y	?	y	y	y	y
6	democrat	n	y	y	n	y	y	n	n	n	n	n	n	y	y	y	y
7	democrat	n	y	n	y	y	y	n	n	n	n	n	n	?	y	y	y
8	republican	n	y	n	y	y	y	n	n	n	n	n	n	y	y	?	y
9	republican	n	y	n	y	y	y	n	n	n	n	n	y	y	y	n	y
10	democrat	y	y	y	n	n	n	y	y	y	n	n	n	n	n	?	?

Figure 2-1-1: a section of raw data

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	0	1	7	1	7	7	7	1	1	1	7	4	7	7	7	1	7
2	0	1	7	1	7	7	7	1	1	1	1	1	7	7	7	1	4
3	1	4	7	7	4	7	7	1	1	1	1	7	1	7	7	1	1
4	1	1	7	7	1	4	7	1	1	1	1	7	1	7	1	1	7
5	1	4	7	7	1	7	7	1	1	1	1	7	4	7	7	7	7
6	1	1	7	7	1	7	7	1	1	1	1	1	1	7	7	7	7
7	1	1	7	1	7	7	7	1	1	1	1	1	1	4	7	7	7
8	0	1	7	1	7	7	7	1	1	1	1	1	1	7	7	4	7
9	0	1	7	1	7	7	7	1	1	1	1	1	7	7	7	1	7
10	1	7	7	7	1	1	1	7	7	7	1	1	1	1	1	4	4

Figure 2-1-2: the section of data after pre-processing

2.2 Implementation of a simple artificial neural network:

In this paper, all codes for implementations are based on Python and PyTorch programing. The data set is divided into two sections, one contains 70 percent of data representing for training data and the other 30 percent is testing data. Then two tensors are defined because PyTorch need to work on tensors rather than raw data. A three layers network is initialized, with a 16 nodes input layer, a 10 nodes hidden layer and a 2 nodes output layer. The input layer with 16 nodes receives 16 integers of the data set after pre-processing from index 2 to 17, and finally the output layer will give a prediction of it. The network uses sigmoid function for hidden neurons, back-propagation model, Cross-entropy loss as loss function and Stochastic Gradient Descent method for optimizer. The network is trained 500 times with learning rate as 0.0006. An example code is provided in figure 2-2.

```
# Initial parameters
input_neurons = n_features
hidden_neurons = 10
output_neurons = 2
learning_rate = 0.006
num_epochs = 500
# define a neural network structure with only 3 layers
class ThreeLayerNet(torch.nn.Module):
    def __init__(self, n_input, n_hidden, n_output):
        super(ThreeLayerNet, self).__init__()
        # define the hidden layer
        self.hidden = torch.nn.Linear(n_input, n_hidden)
        # define the output layer
        self.out = torch.nn.Linear(n_hidden, n_output)
```

Figure 2-2 part-1

```
def forward(self, x):
       h_input = self.hidden(x)
       h_output = F.sigmoid(h_input)
       y_pred = self.out(h_output)
       return y_pred
# define a NN using the structure defined above
net      =      ThreeLayerNet(input_neurons,      hidden_neurons,
output_neurons)
# define loss function
loss_func = torch.nn.CrossEntropyLoss()
# define optimiser
optimiser = torch.optim.SGD(net.parameters(), lr=learning_rate)
# store all losses for visualisation and debug
all_losses = []
```

Figure 2-2 part-2

2.3 Evaluation of the neural network:

Two performances are evaluated with the neural network after training 500 times. One is the accuracy of testing data, the other one is time performance. After finish training by using training data set as described in section 2.2, the testing data set is tested by the network and an accuracy of the testing data set will be printed out. The accuracy here means when the testing data is applied in the well-trained data, if the output calculated by network match the actual class (category) in the data set. The time performance is the total time including reading in data, training network and testing with the accuracy of testing data set. A confusion matrix will also be visualized to observe the result of testing data, which can conclude that how many objects in a certain type have been classified into a wrong classification. Figure 2-3 shows an example evaluation output of the neural network.

```
Testing Accuracy: 89.47 %

Confusion matrix for testing:

 42  10
  4  77
[torch.FloatTensor of size 2x2]

total time consuming:  0.24301385879516602
```

Figure 2-3: an example output of evaluation

2.4 An improvement neural network using 10-fold cross-validation:

K-fold cross-validation is a method to improve performance of a neural network. Jung, Y (2018) claims that “K-fold cross-validation (CV) is widely adopted as a model selection criterion. In K-fold CV, K - 1 folds are used for model construction and the hold-out fold is allocated to model validation.”[1]. The K-fold cross-validation method is also mentioned by L K Milne, T D Gedeon and A K Skidmore (1995). In their paper, when trying to solve the geographical problem by using neural network to classify data set with a 3 layer (input layer, single hidden layer, output layer) network, a cross-validation method was applied [2]. This gives me an idea of improving performance of the neural network described in section 2.2.

To apply K-fold cross-validation, first is to determine the value of K. Here I decide K = 10, so the data set after pre-processing is randomly divided into 10 sections with nearly equal size. The validation needs to run the code 10 times and each time 1 section will be selected as testing set, while the rest 9 sections are used as training data. In each iteration of processing, different section is chosen to be testing data so in 10 times of processing no any row of data is used twice as testing data. After 10 times of training and testing, an average accuracy and total time will be calculated to evaluate the performance of the neural network. Figure 2-4-1 shows an example code of how to do the 10-fold cross-validation.

```
K =10,  count = 0,  number_each = int(len(data.index)/K)
# start to use 10-fold cross-validation
for count in range(K):
    # initial test_data and train_data
    test_data = pd.DataFrame()
    train_data = pd.DataFrame()
    # divide test_data and train_data
    if(count == 0):
        test_data = data[count*number_each:(count+1)*number_each]
        train_data = data[(count+1)*number_each:]
    else:
        train_data_one = data[:count*number_each]
        test_data = data[count*number_each:(count+1)*number_each]
        train_data_else = data[(count+1)*number_each:]
        train_data = pd.concat([train_data_else,train_data_one])
```

Figure 2-4-1 10-fold cross-validation implementation example

3. Results and Discussion:

Also after applying the 10-fold cross-validation described in section 3.4, we can get Figure 4 for the relationship among number of hidden neurons, total process time and average accuracy.

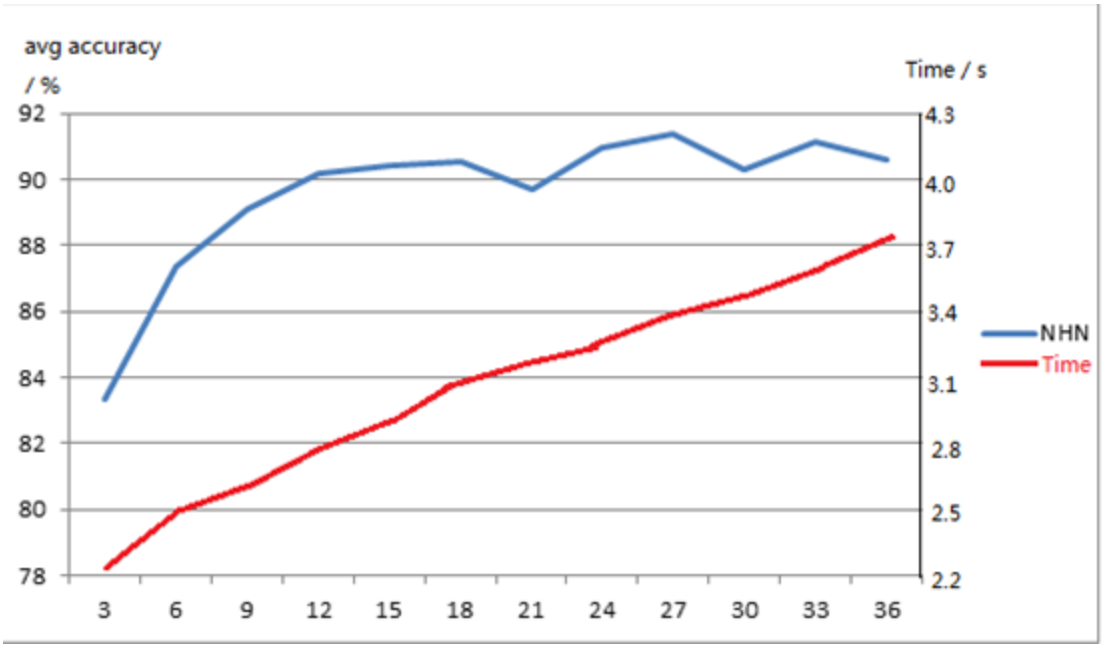

Figure 3: performances with 10-fold cross-validation

As it can be observed in figure 3, with the increasing of numbers of hidden neurons, the average accuracy will first grow rapidly and then be stable in a small range (88.8% to 91.6% in the chart), but the time consuming is keeping increasing because the network is more complex. This gives a recommendation of how to improve my previous neural network, which means 12 to 18 hidden neurons are enough for the only hidden layer while training, because the accuracy is large enough (also cannot be considerably better) and will not consume too much time.

As for my previous unmodified three layers network, it has 10 hidden neurons in the hidden layer, and performs the average accuracy of 89.32 percent, so if I modified it as the recommendation suggests, with 15 hidden neurons in the hidden layer, the accuracy will increase a little bit and will cost a little bit more time. The K-fold cross-validation helps to give a better performance with a neural network.
Bonet B and Geffner H (1998) had used the same data set in their research paper [3]. The accuracy in their research paper is significantly higher than the testing accuracy from my network. Some difference between their method and mine can be discussed. First is the pre-processing for the data set. Bonet B and Geffner H (1998) used a different method to pre-process the categorical values and missing values, which is more reasonable than the pre-processing method as described in section 2.1 (So it can also be indicated that simply apply the missing value as an average value of "1" stand for "n" and "7" stand for "y" is not suitable). The second different is the training iterations. In the method for my paper, all training data has only been trained 500 times, which is less than Bonet B and Geffner H (1998) did, which might cause the network not well trained enough. For the third one, the hierarchy of my artificial neural network could be excessively simple, which just has one layer of hidden neurons.

4. Conclusion and Future Work:

In this paper, I aim to solve a classification problem for a real world data set with artificial neural network implemented by Python / PyTorch coding and discuss of the performance. It is obviously that this neural network is still not a good classification tools for this dataset although it has almost 90% accuracy when testing with testing data set. To improve the behavior of the classification network, many of works can be done in the future. The pre-processing of data set can be handled in a better way, for example, "y", "n", and missing values in the original data set can be changed into more reasonable integers rather than "7", "1" and "4", and some outliers' data (incorrect data recording when generating the data set) should be detected and modified (usually should be cancelled). Also the hierarchy of neural network can be modified into a more complex one (i.e. more hidden layers with more hidden neurons). The parameters of the network, especially the number of training times, learning rate, hidden neuron functions and optimizers, should also be adjusted into some new ones so as to receive a better performance as for both accuracy and time consuming. There are still variables of methods remaining to improve the classification network, and some methods can be applied on it in the future.

5. References:

1. Jung Y. Multiple predicting K-fold cross-validation for model selection. Journal of Nonparametric Statistics. 2018;30:197.
2. L K Milne, T D Gedeon, A K Skidmore, "Classifying Dry Sclerophyll Forest from Augmented Satellite Data: Comparing Neural Network Decision Tree & Maximum Likelihood", Proceedings of the Australian Conference on Neural Networks, pp. 160-163, 1995.
3. Bonet B, Geffner H. Learning Sorting and Decision Trees with POMDPs[C]//ICML. 1998: 73-81.