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

Yukun, Hao
Research School of Computer Science, Australian National University
U6013736@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 and a feature selection technique based on genetic algorithm. 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: Artificial Neural Network, K-fold Cross-validation, Real World Data Set, Pre-processing of data, Feature selection, Genetic Algorithm

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. And then, a genetic algorithm about feature selection will be discussed to improve the performance of the network. 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 "1". For those attributes from index 2 to 17 in figure 1-2, all "y" are read as integer "7", while "1" for "n" and an unique value for "?" in different columns ("?" stand for missing values in the raw data set). The actual values for missing value in each column are decided by the average value of all "7" and "1" in the column. 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, and also “1” for “n” is because I still want this input provide some information when training an artificial neural network rather than simply gives a value “0” which might not give any comment to the network. As for missing value “?”, it indicates that if this column has more instances values “7” (or “y”), then the missing value should more likely to be “y” rather than “n”, and the missing value in this column should be larger than “4”. 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	3.02	7	7	7	1	7
2	0	1	7	1	7	7	7	1	1	1	1	1	7	7	7	1	4.47
3	1	3.55	7	7	3.42	7	7	1	1	1	1	7	1	7	7	1	1
4	1	1	7	7	1	3.89	7	1	1	1	1	7	1	7	1	1	7
5	1	7	7	7	1	7	7	1	1	1	1	7	3.29	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	3.83	7	7	7
8	0	1	7	1	7	7	7	1	1	1	1	1	1	7	7	3.34	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	3.34	4.47

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
```

Figure 2-2 part-1

```
# 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)
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

2.5 An improvement of the neural network using genetic algorithms for feature selection:

Genetic algorithm, as a kind of bio-inspired technology that simulate from evolution, is used widely to find optimal solutions in some difficult problems. According to Yang J. and Honavar V. (1998), “Pattern classification and knowledge discovery problems require selection of a subset of features to represent the patterns to be classified. This is due to the fact that the performance of the classifier and the cost of classification are sensitive to the choice of the features used to construct the classifier. Genetic algorithms offer an attractive approach to find near-optimal solutions to such optimization problems.”[4]. More specifically, a genetic algorithm, gives a recommendation of which of those features showed in figure 1-2 should be used as inputs, will be applied for feature selection to improve the time performance as well as accuracy for the artificial neural network discussed in section 2.2.

First step of applying the genetic algorithm is to define chromosome and initialize population. The chromosome of this problem here should be a 16 bits (because there are maximum 16 inputs in the data set) binary string which represents that which features should be used. (i.e. A chromosome

[1,0,0,0,0,0,0,0,0,0,0,0,0,0,1] means that only the first and the last feature should be used to train and test the artificial neural network as described in section 2.2) And a group of population (30 chromosomes in this case) with multiple individuals (chromosomes) will be randomly created as the first generation.

The second step is to define a fitness function that evaluates the performance of each individual in the population. To assess each chromosome, a fitness function that receives a chromosome as input and train a 3-layer artificial neural network containing n inputs (as described in the chromosome), 18 hidden neurons (enough but not too large to save training time) and 2 output neurons is defined, and it will give an accuracy value which used 5-fold cross validation to get average number as output. Each fitness function will train the neural network 500 times and the learning rate is 0.006. After evaluation, a floating point value will be returned to represent if the individual is good or not, basically larger value means the individual is better.

The third step needs to create a crossover function. The concept of crossover also comes from the true world when two or more parents produce offspring by recombining their own genes. In this case, two parents are selected to produce one offspring at one time, and the individual has both some part of gene from both of its parents.

The following step is mutation, which also is a bio-inspired concept from the real world. For each new offspring, after gaining DNA from its parents, it would has a chance (normally the chance is limited) to change itself so as to introduce new genetic part into the population, and makes it possible to perform better in fitness function.

After all of 4 steps described above, a now population will be created with the same number of individuals (chromosomes) as the old population, and all individuals in old population should be deleted, and will never be considered in the following process. And the genetic algorithm will repeat from step 2 to step 4 until the 60 generation is created, which will give a final result for the feature selection problem.

After applying the genetic algorithm, a fixed length (16 bits here) binary string will be received from it and figure 2-5-1 is an example of the binary string.

Figure 2-5-1 an example of a result received from feature selection genetic algorithms

In figure 2-5-1, the "Most fitted DNA" indicates that the 1st, 4th, 6th, 9th, 10th, 16th features of the data set described in section 1 of the paper should be used for the classifier, and this is an example suggestion of how we can choose some significant meaningful features from a real world data set to improve the accuracy and time performance of the existing artificial neural network.

3. Results and Discussion:

Also after applying the 10-fold cross-validation described in section 2.4, we can get Figure 3-1 for the relationship among number of hidden neurons, total process time and average accuracy, which can give a good suggestion of how the existing artificial neural network should be improved as for its number of hidden neurons.

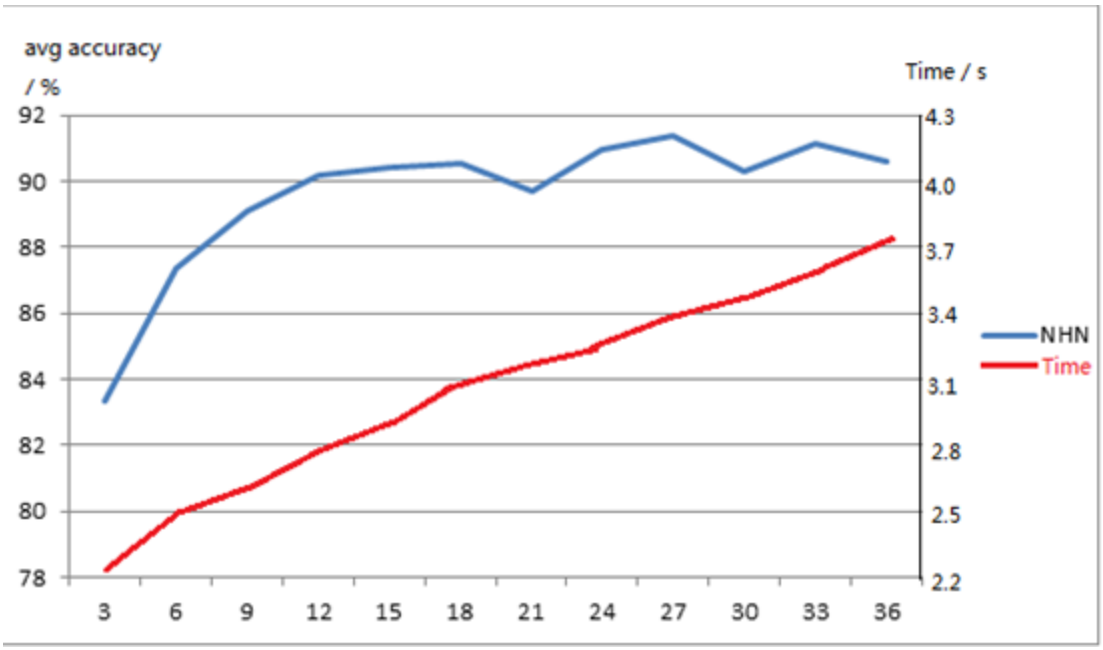

Figure 3-1: performances with 10-fold cross-validation

As it can be observed in figure 3-1, 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.

After applying the genetic algorithm described in section 2.5, figure 3-2 shows an example of how it works with 60 generations.

```
In generation:  1
Most fitted DNA:  [1 0 1 1 1 1 0 0 0 0 1 1 0 0 0 0]
In generation:  9
Most fitted DNA:  [1 0 1 0 1 1 0 0 0 0 1 1 0 0 0 0]
In generation:  29
Most fitted DNA:  [1 1 1 0 1 1 0 0 0 0 1 0 0 0 0 0]
In generation:  60
Most fitted DNA:  [1 1 1 1 1 1 0 0 0 0 1 1 0 0 1 0]
```

Figure 3-2: an example performance of the described genetic algorithm

As it can be seen in figure 3-2, in different generation, the most fitted DNA can be different, which means the algorithm is working to solve the optimize problem. From the generation 60, the result we

can receive from the algorithm is "Most fitted DNA: [1,1,1,1,1,1,0,0,0,0,1,1,0,0,1,0]", which indicates that for the artificial neural network mentioned in section 2.2 and the data set described above, select the 1st, 2nd, 3rd, 4th, 5th, 6th, 11st, 12nd, 15th features are most significantly to do the classification job. And figure 3-3 is the performance with the same 10-fold cross-validation using those features to train and test the neural network.

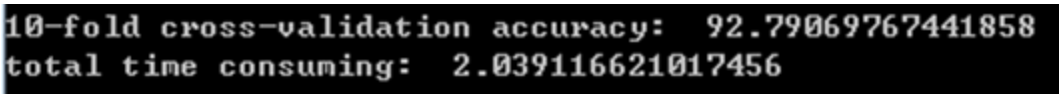

Figure 3-3: an example performance using selected features to train and test the same network using 10-fold cross-validation

Comparing with figure 3-1 when number of hidden neurons is 18, the performance of both accuracy and time are better, which means with less significant features being treated as inputs, the neural network will spend less training time and the accuracy is higher, and that is the main object of applying an feature selection based on genetic algorithm using for an existing artificial 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. However, I have discussed some techniques of feature selection based on genetic algorithms, which Bonet B and Geffner H (1998) did not mention in their work.

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 a floating point number around "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.
The feature selection technique mentioned in section 2-5 improves the performance of the network, but some problems of the technique still exist. The first problem is about the fitness function, because I use

the artificial neural network itself as a fitness function for this genetic algorithm, it is not always give a certain fitness value for a certain input. For example, for a given input (chromosome) "[1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1]", the fitness function can give a feedback from "82.79" to "84.19", so an individual could not be the best individual actually even the fitness function suggests so. A possible solution to solve the problem is to test an individual and get an average value as its fitness value which indicates that if this individual is good or not, but it will cost more time to implement the genetic algorithm because of training more artificial neural networks. The second problem of the feature selection technique is about its crossover and mutation functions. There are variables of methods can be selected to complete the algorithm and I have just chosen a simple crossover and mutation methods to apply, which are possibly not the best choice. And also, parameters for the algorithm are important, including the population size, mutation rate, cross rate, generation numbers and so on, but it should be tried more times to find a reasonable value (I make the population size as 30 and generation numbers as 60 to save processing time, but certainly the accuracy performance could be affected because parameters are not large enough to explore all possible search place). Another problem should be considered is about the selection of population after reproduction. What I have done is to replace all individuals in the old population with all offspring produced by them, but also as I mentioned, it might not be the best way to select the population. As a result, some other methods of replacement should be considered. The last problem is the total time consuming of the genetic algorithm. It takes more than 30 minutes to finish with a population size of 30 and generation numbers of 60, which is acceptable. But if I want to use larger population with more generations, it could take more time to process. So here is a common idea to reduce the time consuming, using parallel programming. Making all individuals in one generation to test their own fitness value would save plenty of time, for the training and testing neural network part in the fitness function dominates the time performance of the genetic algorithm described in section 2.5.

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.
4. Yang J., Honavar V. (1998) Feature Subset Selection Using a Genetic Algorithm. In: Liu H., Motoda H. (eds) Feature Extraction, Construction and Selection. The Springer International Series in Engineering and Computer Science, vol 453. Springer, Boston, MA.