CLASSIFYING INCOME POTENTIAL FROM THE ADULT DATASET: COMPARING DIFFERENT UNDERSTANDING AND PREPROCESSING DATA

Hongzheng He
Research School of Computer Science, Australian National University
u6342036@anu.edu.au

Abstract. Adult dataset has 48842 instances and 15 attributes. The last one is the qualitative attribute which is called 'income' in the code. My job is to predict each adult's income potential and classify them according to the quantitative attributes. There are different methods and thoughts about the raw data (Reducing the training set, normalizing, stratified sampling and deal with the categorical columns) influencing the final predicting results which will be shown and compared in this paper. Some methods can improve the neural network but some are not absolutely can, and the best result is better than those shown in the paper for the same dataset I choose (Vidya, Sejal and Ronit).

1. Introduction

1.1 Dataset

The dataset I choose is the adult dataset, also called "Census Income" dataset. There are 48842 instances with 14 quantitative attributes and 1 qualitative attribute which all clearly describing its meaning. 14 quantitative attributes: 'age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',' occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss', 'hours-per-week', 'native-country'. Qualitative attribute: 'income'. 'income' has two values which is '<=50k' (less or equal to 50k/yr) and '>50k' (more than 50k/yr). The quantitative attributes are the features and the qualitative attribute is the target.

There are 9 attributes are 'string' within raw data: 'workclass', 'education', 'marital-status', 'occupation', 'relationship', 'race', 'sex',' native-country' and 'income' which are must be converted to numeric in order to let network learn. Thus, each different value of these attributes is arranged as a number. Particularly, '<=50k' 'income' is 0 and '>50k' is 1. The related code is shown below:

```
#convert string to numeric, income:<=50K: 0, >50K: 1
category_col =['workclass', 'race', 'education','marital-status', 'occupation','relationship', 'sex', 'native-country','income']
for col in category_col:
    b, c = np.unique(data[col], return_inverse=True)    data[col] = c
```

The reason I choose this dataset is there is an easy-understanding classification target: predicting the income and classify it into two groups based on the learning on the quantitative attributes. The quantitative attributes varying from different aspects and can be handled with different ways which can increase the differences of preprocess methods.

1.2 Model

The model is a two-layers neural network with 50 hidden neurons as one hidden layer with 1000 epochs. At first, Stochastic Gradient Descent (SGD) is chosen as the optimizer but the result of confusion matrix (fig.1) and the output (fig.2) of each epoch is weird as shown below:

Remark.2 Loss and accuracy are calculated every 50 epochs during the network training. The accuracy means the ratio of the correctly predicted targets to the total targets. The accuracy should increase in ideal situation.

Remark.1 If the actual value equals to the predicted value, the value locating on diagonal will be added. So the number on the diagonal should be larger in ideal situation

```
Confusion matrix for training:

 18123       0
  6006       0
```

Fig.1 Confusion matrix of the trained neural network with SGD optimizer.

```
Epoch [1/1000] Loss: 0.6713  Accuracy: 75.08 %
Epoch [51/1000] Loss: 0.6179  Accuracy: 75.11 %
Epoch [101/1000] Loss: 0.5921  Accuracy: 75.11 %
Epoch [151/1000] Loss: 0.5792  Accuracy: 75.11 %
Epoch [201/1000] Loss: 0.5726  Accuracy: 75.11 %
Epoch [251/1000] Loss: 0.5690  Accuracy: 75.11 %
Epoch [301/1000] Loss: 0.5671  Accuracy: 75.11 %
Epoch [351/1000] Loss: 0.5661  Accuracy: 75.11 %
Epoch [401/1000] Loss: 0.5655  Accuracy: 75.11 %
Epoch [451/1000] Loss: 0.5651  Accuracy: 75.11 %
Epoch [501/1000] Loss: 0.5649  Accuracy: 75.11 %
Epoch [551/1000] Loss: 0.5648  Accuracy: 75.11 %
Epoch [601/1000] Loss: 0.5647  Accuracy: 75.11 %
Epoch [651/1000] Loss: 0.5646  Accuracy: 75.11 %
Epoch [701/1000] Loss: 0.5646  Accuracy: 75.11 %
Epoch [751/1000] Loss: 0.5645  Accuracy: 75.11 %
Epoch [801/1000] Loss: 0.5645  Accuracy: 75.11 %
Epoch [851/1000] Loss: 0.5644  Accuracy: 75.11 %
Epoch [901/1000] Loss: 0.5644  Accuracy: 75.11 %
Epoch [951/1000] Loss: 0.5643  Accuracy: 75.11 %
```

Fig.2 Loss and Accuracy on training set in epochs using neural network with SGD optimizer

I think more data should be lied on the diagonal so the matrix seems to be wrong. And the accuracy stays still so there may be an overfitting condition. Therefore, I choose Rprop optimizer and the result seems to be fine.
According to the paper (L.K. Milne1..., 1995), the raw data always need to be preprocessed to get better predication result. Cumulative histogram enhancement technique (Richards, 1986) is used in the paper (L.K. Milne1..., 1995) which can scale the data to improve the learning effect.

2. Method

2.1 Deleting the incomplete data

After observing the raw data, it's found that there are many instances with incomplete data: 1836 instances with incomplete 'workclass' (5.64%), 1843 instances with null 'occupation' (5.66%) and 583 instances with null 'native-country' (1.79%). These incomplete data will influence the network learning accuracy and waste the learning time. Therefore, deleting the incomplete data is necessary. Fortunately, the incomplete data in adult dataset are not too many, otherwise, the network learning few data is unreliable. In this way, either only the relative complete data is used or the data should be collected again. The related code is shown below:

```
#check the incomplete data
col_names = data.columns
num_data = data.shape[0]
for c in col_names:
    num_non = data[c].isin(["?"]).sum()
    if num_non > 0:
        print (c) print (num_non) print ("{0:.2f}%".format(float(num_non) / num_data * 100)) print ("\n")
#delete the incomplete data
data = data[data["workclass"] != "?"] data = data[data["occupation"] != "?"] data = data[data["native-country"] != "?"]
```

2.2 Normalize

After observing the raw data, the range of each attribute can be much different. In order to avoid the attributes with large range influence overwhelmingly on the learning process than the attributes with little range and make the computing on learning easier, normalizing the data is a useful and powerful method. The related code is shown below:

```
for column in data:
    # only normalizing the features
    if column != 'income':
        data[column] = data.loc[:, [column]].apply(lambda x: (x - x.min()) / (x.max()- x.min()))
```

2.3 Stratified sampling

In the above steps, training set and testing set are respectively chosen random 80% and 20% data. But it's possible that more data belonging to one class are chosen than another class which makes the network learn much feature of one

class. As a result, it's better to force the ratio of data belonging to each class to be the same, in my approach, 80% of both '<=50k' and '>50k' data are training set and the 20% as testing set. The related code is shown below:

```
high_income = data[data['income'] == 1]
low_income = data[data['income'] == 0]
train_data = pd.concat([high_income.sample(frac=0.8, random_state=1),low_income.sample(frac=0.8, random_state=1)])
test_data=pd.concat([high_income.sample(frac=0.2, random_state=1),low_income.sample(frac=0.2, random_state=1)])
```

2.4 Re-arrange data

It's not hard to find some raw data are too detailed, like 'marital-status' and 'workclass'. I guess this may make the network to obtain too much useless different patterns. Then I tried to re-arrange these data to make them more simple and meaningful. For 'marital-status', 'Divorced', 'Never-married', 'Separated' and 'Widowed' all can be defined as 'no married'. 'Married-AF-spouse', 'Married-civ-spouse' and 'Married-spouse-absent' can be defined as 'married'. For 'workclass', 'Self-emp-not-inc' and 'Self-emp-inc' can be defined as 'inc'. 'Local-gov', 'State-gov' and 'Without-pay' can be defined as 'gov'. The related code is shown below:

```
data.replace(['Divorced', 'Married-AF-spouse', 'Married-civ-spouse', 'Married-spouse-absent', 'Never-married','Separated','Widowed'],
        ['not married','married','married','married','not married','not married','not married'], inplace = True)
data.replace(['Private', 'Self-emp-not-inc', 'Self-emp-inc', 'Federal-gov', 'Local-gov','State-gov','Without-pay', 'Never-worked'],
        ['Private','inc','inc','gov','gov','gov','Without-pay','Never-worked'], inplace = True)
```

3. Results and discussion

Firstly, the comparisons of my different methods are shown in the fig.3 :

	Testing Accuracy:	Training time	Testing time
Raw data	75.85 %	95.2651 Seconds	0.0505 Seconds
Incomplete data deleted	81.13 %	87.5845 Seconds	0.0444 Seconds
Incomplete data deleted + Normalize	85.25 %	97.2928 Seconds	0.0413 Seconds
Incomplete data deleted + Normalize + Stratified sampling	85.96 %	109.5549 Seconds	0.0443 Seconds
Incomplete data deleted + Normalize + Stratified sampling + Re-arrange	86.18 %	107.4504 Seconds	0.0417 Seconds

Fig.3 Comparison on accuracy on testing set, training time and testing time of neural networks with different preprocess methods.

It can be seen from the figure, deleting incomplete data can not only improve the testing accuracy but also decrease the training time which means deleting incomplete data can help network learn and save the time. Added normalizing, the testing accuracy is improved obviously which showing normalizing can help the network learning effect. On the other hand, the training time increases, in my opinion, there is no doubt normalizing costs more computing time. However, the testing time decreases, I think because the normalized data does help to save time but if the large size influences. As the figure shows, stratified sampling can help improve the accuracy but not much, I think it is because that the distribution of different target class data in this dataset is almost average, the benefit of stratified sampling is not obvious. About the re-arrange, there is improvement both in testing accuracy and training/testing time. Therefore, the data in adult dataset are truly too detailed.

Next, the accuracy on testing set of neural network with two layers is 0.850 which is a little below than mine (0.862).

The accuracy of other paper (Vidya, Sejal and Ronit) is shown below:

Model	*Accuracy on Training Set*	*Accuracy on Test Set*
Logistic Regression	0.7908386	0.792619203
Extra Tree Classifier	0.7908386	0.821227956
Decision Tree Classifier	0.99996742	0.803396473
K - Nearest Neighbor Classifier	0.99996742	0.795754409
SVM	0.963966899	0.748856956
Gradient Boosting Classifier	0.864012511	0.862900065
Stepwise Logistic Regression	0.848	0.847
Naive Bayes	0.811	0.809
ANN 1	0.847	0.847
ANN 2	0.852	0.850
ANN 3	0.855	0.852
ANN 4	0.856	0.853
ANN 5	0.862	0.853
ANN 6	0.862	0.855

Table 5. Accuracies for Training and Testing Dataset

In addition, after running multiple times, I find the outcome of my code will change but the overall tendency is almost as the same. Stratified sampling may not help sometimes, the figure.1 shows the best result I got.

4. Conclusion and future work

Different methods have been implemented and the results have been compared. Raw data from real-world can be hard to learnt by neural network, it's necessary to preprocess the raw data. And better preprocess depends on better understanding about the data. Deleting incomplete data, normalize and re-arranging both can help improve the effect and effectiveness but re-arranging need to depend on specific dataset and attribute's meaning.

There are still many things need to improve, only two detailed categorical attributes are re-arranged but there are actually more. The cumulative histogram enhancement technique can be implemented to preprocess the data better. And the improvement between epochs is little, I believe there are some hidden connection can be removed to save time. Some kind of network reduction technique can be implemented, like finding the classes of excess units and removing them. (T.D. Gedeon & D. Harris, 1991)

References:

1. Vidya Chockalingam, Sejal Shah, Ronit Shaw : Income Classification using Adult Census Data (publish time not found)

2. L.K. Milne1, T.D. Gedeon1 and A.K. Skidmore2 : CLASSIFYING DRY SCLEROPHYLL FOREST FROM AUGMENTED SATELLITE DATA: COMPARING NEURAL NETWORK, DECISION TREE & MAXIMUM LIKELIHOOD (1995)

3. T.D. Gedeon & D. Harris : NETWORK REDUCTION TECHNIQUES (1991)